ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación: INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

"Desarrollo de un DSS con criterios medioambientales para el diseño de rutas transpirenaicas sostenibles en Navarra"

ALUMNO: Luis Iñaki Torres Valencia

TUTOR: Javier Faulín Fajardo

Pamplona, 15 de Septiembre de 2010

Índice del Proyecto:

ORIGEN Y MOTIVACIÓN

El presente Proyecto Final de Carrera surge como consecuencia y continuación de un proyecto de investigación, realizado por los departamentos de Economía y de Estadística e Investigación Operativa de la Universidad Pública de Navarra, para el Ministerio de Educación y Ciencia. El título del proyecto es "GESTIÓN DEL TRANSPORTE SOSTENIBLE POR CARRETERA EN ZONAS TRANSFRONTERIZAS PIRENAICAS" o como se ha llamado dentro del grupo de trabajo, "TRANSPIR".

El objetivo fundamental de dicho proyecto es de carácter doble:

1. Ofrecer una evaluación económica de las externalidades medioambientales (ruido y emisiones) en zonas transfronterizas pirenaicas.

2. Establecer un análisis conjunto de las actividades de transporte sostenible mediante la optimización de rutas que tengan criterios medioambientales verificados con datos de campo reales.

Como complemento a este último punto, se pensó en diseñar un Decision Support System (DSS) para el cálculo de rutas más respetuosas y coherentes con los diferentes criterios medioambientales seleccionados.

Este es el motivo por el cual, el presente Proyecto de Fin de Carrera, consiste precisamente en el estudio, diseño y desarrollo de un DSS para el cálculo de rutas con criterios medioambientales y sociales, aprovechando la experiencia y conocimiento adquiridos por el autor durante su participación en el proyecto "TRANSPIR".

Por otro lado, de manera independiente del proyecto "TRANSPIR", se pensó que la idea del DSS con criterios medioambientales para cálculo de rutas puede ser una forma de avanzar en el conocimiento de un nuevo campo, el de un transporte sostenible, que en el momento en el que nos encontramos, es uno de los temas de interés de los organismos gubernamentales de las principales potencias mundiales así como de las empresas relacionadas con el sector.

En el caso de Navarra, tanto el gobierno Foral, como el estado español y sobretodo la Unión Europea, están legislando y proponiendo nuevas medidas con el objetivo de reducir los efectos negativos que las emisiones procedentes del transporte provocan en la población y en el medioambiente.

OBJETIVOS Y CARACTERÍSTICAS DEL PROYECTO

El objeto principal de este proyecto consiste en el diseño y desarrollo de un sistema de apoyo a la decisión (**D**ecision **S**upport **S**ystem) que teniendo en cuenta los factores de contaminación acústica y del aire, proponga rutas alternativas para el transporte, intentando minimizar tanto el número de personas como las zonas de interés ecológico afectadas por dichos factores.

En general, podemos decir que un DSS es un sistema informático utilizado para servir de apoyo en el proceso de toma de decisiones. La decisión es una elección entre alternativas basadas en estimaciones de los valores de esas alternativas. El apoyo a una decisión significa ayudar a las personas que trabajan individualmente o en grupo a reunir inteligencia, generar alternativas y tomar decisiones.

El modo en el que trabaja el DSS da como resultado un grupo de soluciones distintas para cada problema. Cada una de las soluciones ofrecidas por el DSS deberá satisfacer una serie de restricciones de carácter medioambiental y social preestablecidas. Finalmente, esto nos permitirá analizar, comparar y escoger aquella solución que más nos interese en cada caso.

La idea inicial se centraba en el problema del transporte transpirenaico en Navarra, proponiendo un DSS para encontrar las rutas menos perjudiciales tanto en el medio como en la población de Navarra. Sin embargo, el DSS diseñado en este proyecto funciona independientemente del ámbito geográfico o de las características orográficas del problema. Cualquier problema puede ser procesado por el DSS, siempre y cuando se siga el diseño de la estructura del problema y se cumplan los requisitos funcionales del programa.

El hecho de no estar limitado solo al ámbito de la comunidad Foral de Navarra dota de un valor añadido al DSS, haciendo que sea posible su reutilización y adaptación a problemas similares en cualquier otro lugar donde exista una red de carreteras y un volumen de tráfico que transite por ellas.

El DSS ha de mostrar gráficamente las soluciones obtenidas para cada problema en concreto. De cara al futuro, puede ser interesante avanzar en el diseño del DSS, buscando el modo de compatibilizarlo con herramientas GIS o con complementos web de uso extendido como GOOGLE MAPS.

Las rutas obtenidas mediante este DSS han de guardar un compromiso entre el coste final y los daños causados a terceros. El equilibrio de esta ecuación está en función de los requerimientos del usuario final, pudiendo ser este un particular o el encargado de una empresa de transportes.

1. INTRODUCCIÓN AL VEHICLE ROUTING PROBLEM. ALGORITMOS.

Para poder entender los algoritmos de cálculo de rutas, es necesario comprender primero el problema para el que están diseñados, así como sus posibles variantes. Por este motivo se va a hacer una breve introducción al problema del cálculo de rutas para vehículos, más conocido por sus siglas en inglés, VRP (Vehicle Routing Problem).

El objetivo del VRP (*Toth P. y Vigo D.*, 2002) es entregar bienes a un conjunto de clientes que normalmente se encuentran dispersos en una zona geográfica, minimizando el coste.

Las principales características del problema son las siguientes:

1. Las demandas de cada uno de los clientes son conocidas previamente, excepto en la variante del VRP con demandas estocásticas.

2. La entrega debe realizarse con el mínimo costo, encontrando para ello las rutas óptimas que se originan y terminan en el almacén o depósito.

Todos los clientes deben ser atendidos una sola vez, por lo que a cada uno le visitará un vehículo con una capacidad de carga mayor que la demanda del cliente.

Existen tres componentes comunes a todas las variantes del VRP:

- **Los clientes:**

  Cada cliente tiene una demanda que deberá ser satisfecha por algún vehículo. En la mayoría de los casos, la demanda es un bien o producto que ocupa un volumen en los vehículos. Sin embargo, existen casos en los que la demanda no es un producto físico, sino un servicio, por lo que el vehículo simplemente ha de visitar al cliente.

  Como se verá más adelante, existen variantes del VRP donde los clientes imponen restricciones tales como el horario en el que pueden ser atendidos, lo que origina ventanas de tiempo asociadas a cada cliente.

- **El almacén o depósito:**

  Tanto los vehículos como los bienes a distribuir suelen estar ubicados en almacenes o depósitos.

  Normalmente las rutas de reparto comienzan y terminan en el almacén. Pueden existir casos con múltiples almacenes, cada uno de ellos con una flota de vehículos asignada.

  Al igual que los clientes, las bodegas también pueden tener ventanas de tiempo asociadas, ya que en algunos casos se debe considerar el tiempo necesario para cargar o preparar el vehículo antes de comenzar su ruta, o simplemente se quiere evitar una congestión de vehículos en el almacén.

- **Los vehículos:**

  La capacidad de un vehículo puede estar expresada en peso, volumen, número de clientes, entre otras. En algunos casos se desea que la cantidad de trabajo realizado por los vehículos no sea muy dispar.

  En general se asume que cada vehículo recorre una sola ruta en el periodo de planificación, pero últimamente se han estudiado modelos en los que un mismo vehículo puede recorrer más de una ruta.

El VRP en la práctica.

Es evidente que el VRP tiene una gran relevancia en práctica, ya que el transporte de mercancías se ha convertido a lo largo de los últimos siglos en una necesidad de la sociedad moderna y está presente en la mayoría de las actividades económicas y de ocio.

De hecho, uno de los factores que más influyen en el desarrollo de una región es la calidad de sus redes de transporte, tanto carreteras o infraestructuras ferroviarias, como puertos marítimos y aeropuertos. Las regiones mejor comunicadas, normalmente han sufrido un mayor desarrollo industrial y económico, y esto es debido a que, en general, una buena red de transporte favorece y facilita las actividades industriales.

Desde el punto de vista del mundo empresarial, el VRP es uno de los retos logísticos más importantes al que se enfrenta, ya que unos de los principales objetivos de toda empresa es encontrar el modo de ahorrar costes, y con una buena planificación de rutas se pueden reducir los gastos derivados del transporte, aumentando de este modo el margen de beneficio. Además, gracias a una buena planificación de rutas se consigue mejorar la eficiencia y la productividad de las actividades afectadas por los tiempos de reparto y carga de mercancías.

Otro de los motivos del continuo estudio del VRP, es que supone un reto debido a la dificultad de resolver dicho problema, ya que se pueden encontrar soluciones óptimas para casos pequeños en un tiempo razonable, pero el problema se complica enormemente al aumentar el tamaño del caso de estudio. Desde un punto de vista matemático, esto se debe a que el VRP pertenece a conjunto de los problemas NP.

Demostrar que un problema es NP-completo equivale a demostrar que no está en P, es decir, que no tiene una solución determinista en un tiempo polinomial (*Golden et al.* 2008). Conocer en qué grupo se encuentra cada problema es muy importante, puesto que permite abandonar la búsqueda de un algoritmo exacto para la solución óptima y centrarse en objetivos realizables como encontrar algoritmos para obtener soluciones aproximadas (heurísticas o metaheurísticas).

Además, en la mayoría de casos prácticos, una buena aproximación resulta tan útil como lo pudiera ser la solución exacta, por lo que para muchos problemas se acepta la mejor aproximación como la solución de ese problema, y no se emplean más esfuerzos ni recursos en la búsqueda de la solución óptima.

Como ya se explica más adelante, en la actualidad se están estudiando variantes del VRP donde los requisitos impuestos hacen que cambie el objetivo fundamental de minimizar el coste de las rutas. En estos casos, la ruta solución no tiene por qué ser necesariamente la más corta, sino aquélla que satisfaga mejor las restricciones. Los algoritmos para solucionar estas variantes tendrán como principal objetivo cumplir las restricciones pero intentarán hacerlo con el mínimo coste posible.

1.1. Variantes del VRP

La idea del VRP original es muy útil para entender las bases del problema, pero en la práctica, los casos reales de problemas del tipo VRP contienen una serie de restricciones y características especiales que los diferencian de otros casos.

A continuación se exponen algunas de las principales variantes de VRP, para cada una de las cuales se han diseñado algoritmos específicos que resuelven el problema cumpliendo sus restricciones propias.

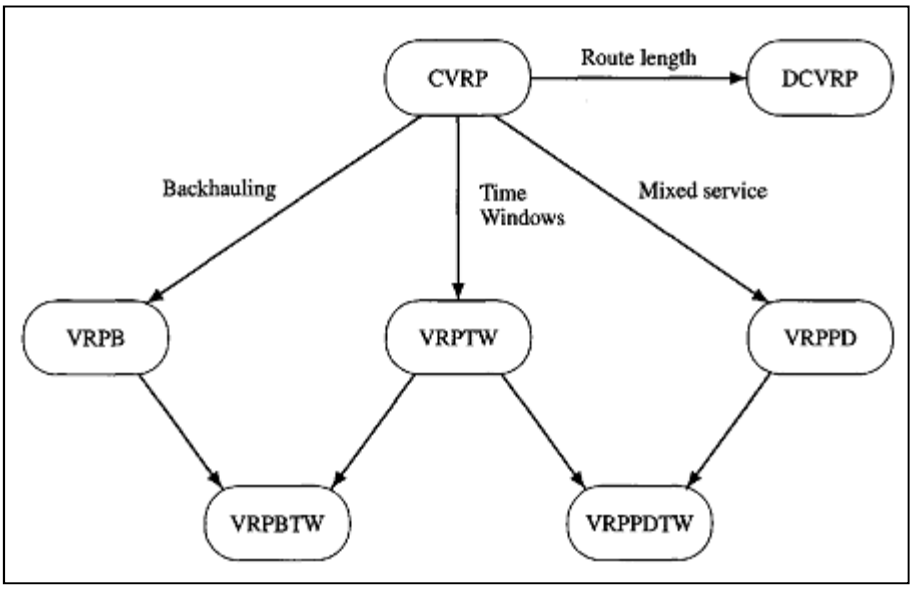


Ilustración 1.1. Variantes del VRP (*The VRP Web*)

Capacited VRP (CVRP)

En ésta variante del VRP existe una flota de vehículos de reparto con una capacidad de carga uniforme que deben atender las demandas conocidas de los clientes con un coste mínimo de transporte.

Es decir, sobre el VRP se impone la restricción de que todos los vehículos tienen una capacidad de carga uniforme de un solo producto.

Multiple Depot VRP (MDVRP)

Existen varios almacenes desde los cuales se puede atender a los clientes. Si los clientes se agrupan en torno a los distintos almacenes, se puede resolver el problema como un conjunto de subproblemas VRP independientes.

El MDVRP requiere que se asigne cada cliente a un almacén, así como conocer el número de vehículos establecidos en cada almacén. Un vehículo parte de un almacén, atiende a los clientes y regresa al punto de partida.

Periodic VRP (PVRP)

En el problema original del VRP, la planificación se realiza para un solo día. En este caso, se generaliza el VRP extendiendo el periodo de planificación a varios días.

Split Delivery VRP (SDVRP)

En el problema original del VRP, los vehículos visitan una sola vez a cada cliente, ya que se asume que la demanda de cada cliente en ningún caso supera a la capacidad del vehículo. En el Split Delivery VRP no existe esta restricción, siendo posible que un mismo cliente sea visitado varias veces por distintos vehículos.

Stochastic VRP (SVRP)

En estos problemas, uno o más de los componentes son aleatorios. Pueden ser tanto los clientes como las demandas o incluso los tiempos de transporte y servicio.

VRP with Backhauls (VRPB)

Es una variante del VRP en el que el cliente puede demandar o devolver mercancías. Se debe tener en cuenta que los vehículos no sólo se van vaciando, ahora pueden ir recogiendo mercancías mientras hacen su ruta.

VRP with Pick-Up and Delivering (VRPPD)

Unas determinadas mercancías necesitan ser recogidas de determinadas localidades para ser repartidas en otras.

El objetivo es encontrar las rutas óptimas en las que los vehículos visiten los puntos de recogida y los puntos de destino.

VRP with Satellite Facilities (VRPSF)

En este caso se estudia la posibilidad de que se pueda reabastecer los vehículos sin necesidad de que retornen a almacén central.

VRP with Time Windows (VRPTW)

En el VRPTW se añade una restricción temporal, es decir, un plazo de tiempo en el cual cada cliente debe ser atendido. Las ventanas de tiempo pueden referirse al horario en el que se puede atender a cada cliente, así como al horario en el que los vehículos pueden ser cargados en el almacén.

En general, cada problema VRP de la vida real supone en sí mismo una variante del problema original, ya que cada caso tiene sus características y restricciones propias. Es por esto que necesitan "adaptarse" los algoritmos existentes al problema concreto.

En este proyecto, la solución que se busca no es aquella que minimiza el coste total de las rutas. La solución esperada debe minimizar el impacto del transporte en la sociedad y en el ecosistema, y como prioridad complementaria, se intentará que además se minimice el coste total.

1.2. Algoritmos para el VRP

Como se ha explicado anteriormente, el VRP está ligado fuertemente a problemas relacionados con el transporte de mercancías, pero aparece en muchos más ámbitos, por lo que es uno de los problemas de optimización más importantes y a la vez más estudiados, para el cual se han ido proponiendo distintas soluciones a lo largo de los últimos 50 años.

En 1959, Dantzing y Ramser presentaron el problema, describiendo una situación real relacionada con el abastecimiento de gasolina a las estaciones de servicio y para la que propusieron la primera formulación matemática para su resolución.

En 1964, G. Clarke y J.M. Wright (*Clarke and Wright*, 1964) propusieron un método heurístico eficaz que mejoraba la solución propuesta por Dantzing y Ramser. Desde entonces, se han propuesto cientos de soluciones para las distintas variantes del VRP. Algunas de éstas propuestas buscan la solución óptima del problema, otras persiguen una aproximación de la solución que sea válida. Todas las metodologías propuestas se pueden clasificar en función del modo en el que se tratan de llegar a la solución, como se ve a continuación.

En este capítulo de introducción, se presentan algunos de los métodos más utilizados para la resolución de estos problemas hasta el momento. Sin embargo, existen continuas mejoras de los algoritmos existentes, creándose así un amplio abanico de soluciones para una misma familia de problemas. Cada una de estas posibilidades tiene sus ventajas e inconvenientes respecto a las demás, ya que cada una tiene unas necesidades de recursos distintas y aporta soluciones en tiempos de cómputo distintos.

Los diferentes algoritmos utilizados para el VRP se pueden dividir en dos grandes familias:

1. Los Heurísticos. Los algoritmos explicados son:
    a. **Clarke & wrigth.**
    b. **Método del barrido.**
    c. **Búsqueda local (Lin y Kernighan)**

2. Los Metaheurísticos. Los algoritmos explicados son:

    a. **GRASP**.

    b. **Algoritmos Genéticos (GA).**

    c. **Búsqueda Tabú**.

    d. **Simulated Annealing (SA).**

1.2.1. Métodos heurísticos

Algoritmo de Clarke & Wright (Método de los ahorros)

Este algoritmo fue diseñado para un problema en el que se asume que existe un depósito central, con uno o varios vehículos de entrega y n localizaciones de clientes, cada uno con un requerimiento conocido previamente. El objetivo es averiguar cómo asignar vehículos a clientes, para cumplir la demanda del cliente y satisfacer las restricciones de minimizar los costos.

G. Clarke y J.M. Wright en "*Scheduling of Vehicles from a Central Depot to a Number of Delivery Points*", discuten una técnica para encontrar buenas rutas. Este algoritmo conocido como de Clark & Wright entrega soluciones aceptables, y es muy utilizado en aplicaciones prácticas, por la corta cantidad de tiempo y la facilidad con que se aplica.

Identificando el depósito como localización 0, y los clientes en las localidades 1, 2, 3, …n, se asume que son conocidos los costes de travesía desde el depósito a cada localidad del cliente; esto es:

Coj = Coste de hacer un viaje desde el depósito hasta el cliente.

Para implementar el método, se hace necesario conocer el costo del viaje entre los clientes, lo que significa que se asume que todos los costes de travesía son conocidos.

Cij = Coste de hacer un viaje desde la localidad i a la localidad j.

Para propósitos prácticos, se considera el caso en el cual Cij = Cji, para todo 1<=i, j<=n.

El método procede del siguiente modo:

Suponer inicialmente que hay un vehículo asignado a la visita de cada cliente. Esta solución inicial consiste de n rutas separadas desde el depósito a cada localidad del cliente.

En consecuencia el coste total del viaje para esta solución inicial es:

$$2\sum_{j-1}^{n} C_{0j}$$

Ahora se supone que se enlazan los clientes i y j. Esto significa que se dirige el vehículo desde el depósito hasta i, y luego hasta j, desde donde se regresa al depósito.

El ahorro realizado en un enlace i a j es:

$$S_{ij} = 2\,(Coi + Coj) - (Coi + Cij + Coj) = Coi + Coj - Cij$$

Este método calcula Sij para todas las posibles parejas de combinaciones de localidades de los clientes i y j, estableciendo un ranking en orden decreciente.

El número total de evaluaciones Sij requeridas viene dado por el número total de combinaciones, tomadas de dos:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

En el siguiente ejemplo se visualiza cómo se construyen rutas con el método de los ahorros:

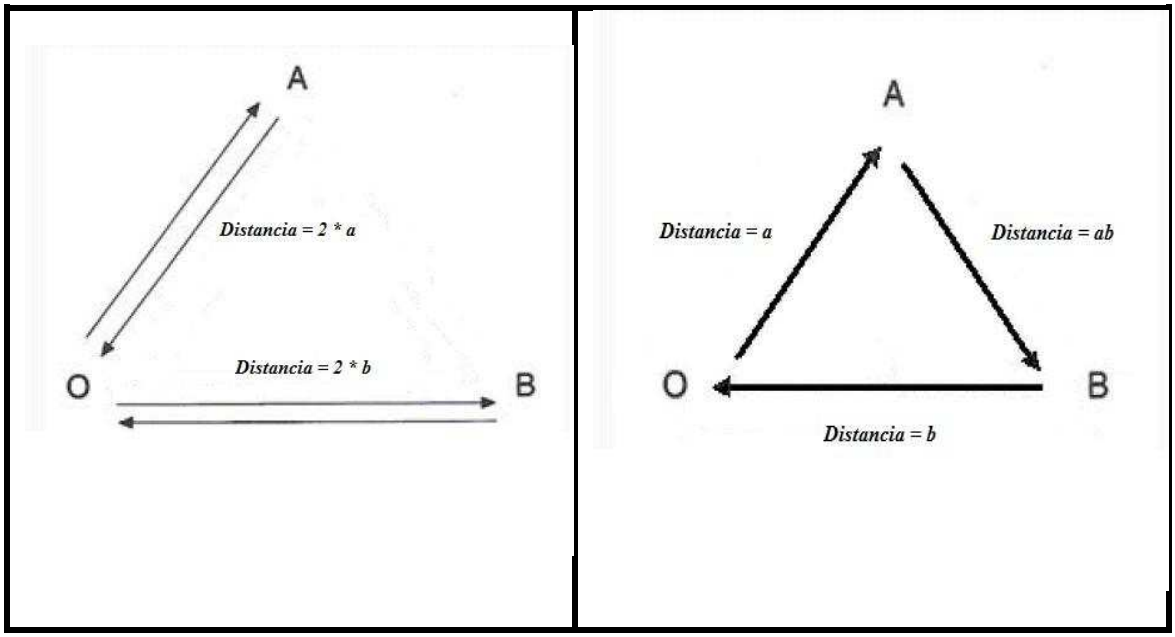


Ilustración 1.2. Método de los ahorros.

En este ejemplo, se inicia el diseño de la ruta como en la figura de la izquierda, es decir, creando rutas de ida y vuelta para cada cliente.

El coste de servir a A y B desde O es la suma de ir y volver de A más ir y volver de B.

Coste 1 = 2*a + 2*b

A continuación, se prueba generando una nueva ruta, en la que unimos A y B.

El coste de esta nueva ruta es igual al coste de visitar A, más el de visitar B desde A, más el coste de volver a O desde B.

Coste 2 = a + ab + b

Si el coste de la segunda ruta (Coste 2) es menor que el coste de la primera (Coste 1), eliminamos la Ruta 1 y nos quedamos con la Ruta 2.

El ahorro conseguido es:

Ahorro = a + b - ab

Pseudocódigo del Algoritmo de los Ahorros

Inicialización Ahorros

1. *Tomar un vértice O∈V como base.*
2. *Establecer los n-1 subtours [(O, v), (v, O)] ∀ v ∈V \{O}*

Mientras *(Queden dos o más subtours)* ***hacer***

Para cada par de subtours:

Calcular el ahorro de unirlos al eliminar en cada uno una de las aristas que lo une con O y conectar los dos vértices asociados.

Unir los dos subtours que produzcan un ahorro mayor.

Fin Mientras

Fin Ahorros

Pseudocódigo 1.1. Algoritmo C&W

Método del Barrido.

Se representan el depósito y los clientes como una serie de puntos sobre el plano, asignándoles coordenadas polares (X,Y), donde X es el ángulo e Y es la distancia en línea recta del cliente al origen.

La idea de este método consiste en hacer girar una semirrecta con origen en el depósito e ir "barriendo" los puntos que representan a los clientes, hasta que las demandas de estos clientes completen la capacidad del vehículo.

De este modo se consiguen varias agrupaciones de clientes, cada una de ellas con una demanda menor o igual que la capacidad del vehículo.

Por último, se emplea un algoritmo (por ejemplo el de los ahorros) para cada agrupación, determinando así el orden en el que serán visitados los clientes de un mismo grupo.

En el siguiente ejemplo se ilustra el modo en el que el método de los barridos forma agrupaciones de clientes con una demanda menor o igual que la capacidad máxima de los vehículos:

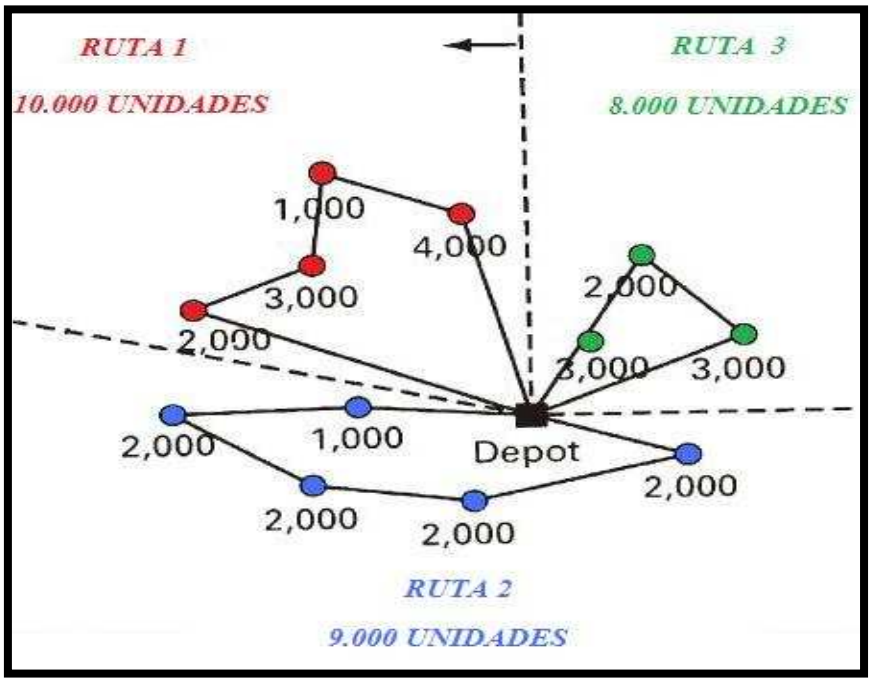


Ilustración 1.3. Método de los barridos. Capacidad máxima de los vehículos: 10.000 unidades.

Técnicas de búsqueda local

Los procedimientos de *búsqueda o mejora local* comienzan con una solución del problema y la mejoran progresivamente. El procedimiento realiza en cada paso un *movimiento* de una solución a otra con mejor valor. El método finaliza cuando, para una solución dada, no existe ninguna solución accesible que la mejore.

El principal problema de los algoritmos de búsqueda local es que suelen quedarse atrapados en un óptimo local.

Para alcanzar una solución mejor a partir de un óptimo local habría que comenzar por realizar movimientos que empeoren el valor de la solución, lo que conduciría a un esquema de búsqueda mucho más complejo, al utilizar el algoritmo tanto movimientos de mejora como de no mejora.

El algoritmo de **Lin y Kernighan** parte de este hecho y propone un movimiento compuesto, en donde pueden existir movimientos simples que no mejoran necesariamente, pero que en su conjunto sí consiguen una mejora respecto a la solución de partida.

De esta forma es como si se realizaran varios movimientos simples consecutivos en donde algunos empeoran y otros mejoran el valor de la solución, pero no se pierde el control sobre el proceso de búsqueda ya que el movimiento completo sí que mejora.

Además, combina diferentes movimientos simples, lo cual es una estrategia que ha producido muy buenos resultados en los algoritmos de búsqueda local. En concreto la estrategia denominada “cadenas de eyección” se basa en encadenar movimientos y ha dado muy buenos resultados en el contexto de la Búsqueda Tabú, explicada más adelante.

Pseudocódigo del algoritmo de Lin y Kernighan

```
Inicialización
        Considerar un ciclo Hamiltoniano inicial
        Movimiento = 1
        Mientras (Movimiento = 1) hacer
                Movimiento = 0
                Etiquetar todos los vértices como no explorados.
                        Mientras (Queden vértices por explorar) hacer
                            1. Seleccionar un vértice i no explorado.
                            2. Examinar todos los movimientos (2-opt, 2-opt, inserción)
                               que incluyan la arista  de i a su sucesor en el ciclo.
                            3. Si alguno de los movimientos examinados reduce la
                               longitud del ciclo, realizar el mejor de todos y hacer
                               Movimiento = 1. En otro caso etiquetar i como
                               explorado.
                        Fin Mientras
        Fin Mientras
Fin Algoritmo
```

Pseudocódigo 1.2. Algoritmo de Lin y Kernighan

1.2.2. Métodos metaheurísticos

Son estrategias para mejorar los procedimientos heurísticos, por lo tanto, el tipo de metaheurística está en función de qué tipo de heurística comprende y se puede hacer una clasificación en:

- **Métodos constructivos.**

  Son los que van incorporando elementos a una estructura inicialmente vacía que representa la solución.

  Un ejemplo es el algoritmo **GRASP**.

- **Métodos Evolutivos.**

  Son métodos que van construyendo un conjunto de soluciones a diferencia de los otros métodos que sólo pasan de una solución a otra en cada iteración.
  El procedimiento consiste en generar, seleccionar, combinar y reemplazar un conjunto de soluciones.

Ejemplos de metaheurísticas evolutivas:

Algoritmos Genéticos, Búsqueda Dispersa (Scatter search).

- **Métodos de búsqueda.**

  Son métodos que presuponen que existe una solución y realizan procedimientos de búsqueda, que no necesariamente encontrarán la solución óptima. Uno de los riesgos al usar un algoritmo de búsqueda es el de alcanzar un óptimo local del que ya no sea posible salir.

  Las principales metaheurísticas de búsqueda global surgen de las tres formas principales de escapar de los óptimos locales:

  1. Volver a comenzar la búsqueda desde otra solución inicial. (*Multi start*)
  2. Modificar la estructura de entornos. (Metaheurística de entornos variables)
  3. Permitir movimientos de empeoramiento de la solución actual. **Búsqueda Tabú** y **Simulated annealing.**

Métodos constructivos

GRASP. Randomized Adaptive Search Procedure

El término GRASP fue introducido (*T.A. Feo and M.G.C. Resende* (1995)) como un método metaheurístico de propósito general. Es un método de en donde cada paso consiste en dos fases, una primera de construcción y otra posterior de mejora.

En la fase de construcción se aplica una heurística constructiva para obtener una solución inicial y en la segunda fase dicha solución se mejora mediante un algoritmo de búsqueda local.

En cada iteración la elección del próximo elemento a ser añadido a la solución parcial está determinada por una función *Greedy,* la cual elige el elemento que da mejor resultado inmediato sin tener en cuenta una perspectiva más amplia.

Se dice que se **adapta** porque en cada iteración se actualizan los beneficios obtenidos al añadir el elemento seleccionado a la solución parcial.

Es **aleatorio** porque no selecciona al mejor candidato sino que, con el objeto de diversificar y no repetir soluciones, se construye una lista de los mejores candidatos, de entre los cuales se elige uno al azar (*Mauricio G. C. Resende. (2008))*

En la **fase de mejora** se realiza un proceso de búsqueda local a partir de la solución construida hasta que no se pueda mejorar más.

El siguiente esquema muestra el funcionamiento global del algoritmo:
Pseudocódigo del GRASP

Mientras** (no se cumpla la condición de parada) **Hacer
Fase Constructiva
1. ***Seleccionar una lista de elementos candidatos.***
2. ***Considerar una lista restringida de los mejores candidatos.***
3. ***Seleccionar un elemento aleatoriamente de la lista restringida.***

Fase de Mejora
1. ***Realizar un proceso de búsqueda local a partir de la solución construida hasta que no se pueda mejorar más.***
2. ***Actualización***
3. ***Si la solución obtenida mejora a la mejor almacenada, actualizarla.***

Fin mientras

Pseudocódigo 1.3. Algoritmo general Greedy Randomized.

Esta metaheurística es una de las más populares debido a su sencillez y facilidad de implementación.

Métodos Evolutivos

Algoritmos genéticos

En los algoritmos genéticos (GA) cada iteración permite obtener un conjunto de soluciones, o poblaciones en curso, y no una única solución en curso. Las soluciones posteriores son obtenidas a partir de parejas constituidas con los elementos de la población y no mediante la transformación de la solución en curso.

El punto de partida de los GA es la teoría de evolución de las especies de Darwin. Si se adopta un procedimiento que dadas dos soluciones genere sucesoras que conserven las mejores características de las misma, la calidad de las soluciones se mantendrá en las generaciones posteriores.

Este algoritmo además puede tener una componente aleatoria, ya que se puede introducir el concepto de "mutación", es decir, modificaciones aleatorias que permiten explorar nuevos caminos en busca de la solución. De éste modo se resuelve la posibilidad de quedar bloqueado en óptimos locales.

El procesamiento matemático de un algoritmo genético simple se muestra a continuación:

Pseudocódigo del Algoritmo Genético:

```
Inicialización
   Población inicial P(0)
   Mientras (no se pasa por todos los clientes) hacer
         Evaluar la idoneidad de cada individuo en P(t)
         Construir una nueva población P (t+1)
            1. Seleccionar elementos de P(t)
            2. Cruzar estos elementos
            3. Mutación de algunos elementos (aleatorio)
   Fin Mientras
  Retornar la mejor solución encontrada
```

Pseudocódigo 1.4. Algoritmo Genético. Ejemplo general.

Métodos de búsqueda

Búsqueda Tabú

La búsqueda tabú (*Toth P. and Vigo D*. (2003))se sirve del concepto de memoria y lo implementa mediante estructuras simples con el objetivo de dirigir la búsqueda teniendo en cuenta la historia de ésta, es decir, el procedimiento trata de extraer información de lo sucedido y actuar en consecuencia.

En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente.

La búsqueda tabú permite moverse a una solución aunque no sea tan buena como la actual, de modo que se pueda escapar de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores.

La búsqueda tabú procede como cualquier algoritmo de búsqueda:

Dada una solución x se define un entorno N(x), se evalúa y se “mueve” a una mejor solución pero, en lugar de considerar todo el entorno o vecindario la búsqueda tabú define el entorno reducido N*(x) como aquellas soluciones disponibles (no tabú) del entorno de x.

A continuación se muestra cómo funciona la búsqueda Tabú. Primero se enumeran en una tabla cada uno de los principales componentes utilizados en este algoritmo:

Componente	Símbolo	Descripción
Función Objetivo	f(x)	La función que nos permite identificar la calidad de la solución, generalmente es maximizar o minimizar una expresión lineal o no lineal.
Espacio de soluciones	S	El conjunto de soluciones posibles, puede estar dado explícitamente o estar dada la estructura de la solución.
Vecindario	N(x)	Las posibles soluciones que pueden seguir a x. En el método simplex serían todos los arreglos tal que una variable actualmente básica pasa a tener valor cero y una no básica pasa a ser básica N(x) →S
Lista Tabú	T(x,k)	Lista de las posibles soluciones que no pueden ser elegidas en la actual iteración. Pueden ser las "L" Soluciones anteriores o los L últimos movimientos.
Conjunto de candidatos	A(x,k)	Soluciones que tienen algún atributo que las hace elegibles aún si estuvieran en la lista Tabú. El conocimiento del experto es importante para determinar qué soluciones pueden aspirar a ser solución en la iteración k.
Número máximo de iteraciones	MAXITER	Para evitar que el algoritmo entre en un proceso sin fin, es aconsejable establecer una cota superior de iteraciones posibles.

Tabla 1. Componentes de la búsqueda Tabú

Para una mejor comprensión de qué es cada uno de estos componentes, a continuación se presenta un resumen del funcionamiento del algoritmo en forma de pseudocódigo:

Pseudocódigo de la Búsqueda Tabú (SIMPLE)

Inicialización

Generar solución inicial x0

k = 1

x = x0 (x es la solución actual)

Mientras** (la condición de finalización no se encuentre) **Hacer

1. ***Identificar N(x) ("Vecindario" de x)***
2. ***Identificar T(x,k) (Lista Tabú)***
3. ***Identificar A(s,k) (Conjunto de "Aspirantes")***
4. ***Determinar N*(x,k) = {N(x) – T(x,k)} A(x,k) ("Vecindario" reducido)***
5. ***Escoger la mejor x de N*(x,k)***
6. ***"Guardar" x si mejora la mejor solución conocida. xk = x***
7. ***Actualizar la lista tabú***
8. ***k := k+1***

Fin Mientras

Pseudocódigo 1.5. Algoritmo de la búsqueda Tabú

Simulated Annealing

El SA (Simulated Annealing) es un método probabilístico que construye nuevas configuraciones aleatoriamente y las somete a reglas de probabilidad para su aceptación, evitando de esta manera la caída en óptimos locales. El proceso termina después de un cierto número de iteraciones y tiene como objetivo obtener soluciones cercanas al óptimo global en problemas de optimización combinatoria compleja.

Este es un método Hill-Climbing y se basa en la analogía con el recocido de los metales que consiste en "calentar" a alta temperatura el sistema que se intenta optimizar, para luego disminuir la temperatura muy lentamente, hasta que ya no ocurren cambios en el sistema. La variación de temperatura en el proceso físico se produce de forma continua, mientras que en el SA sólo puede hacerse escalonadamente.

Pseudocódigo del Simulated Annealing

```
Inicialización
        Solución inicial x
        Temperatura inicial t
        Mientras (no se visitan todos los clientes) hacer
                Seleccionar de forma aleatoria una solución x' próxima a x
                Calcular DC = Coste(x') – Coste(x)
                Si DC<=0 entonces
                        x = x'
                Si DC > 0 entonces
                        x = x' con probabilidad (-DC/t)
                Escoger t = rt (r escalón de temperatura)
        Fin Mientras
Devolver la mejor solución
```

Pseudocódigo 1.6. Algoritmo Simulated Annealing.

1.3. SIMUROUTE

Es un algoritmo que ha sido diseñado para encontrar rutas y soluciones para el VRP introduciendo factores de aleatoriedad, de modo que consigue nuevas rutas que compara con una solución inicial, obtenida mediante el algoritmo de los ahorros (*AA Juan, J Faulin, et al.* 2008).

Una de las ventajas del SimuRoute es que puede proporcionar un gran número de soluciones diferentes, que mantiene una alta calidad, lo que permite realizar comparativas multi-criterio entre ellas, haciendo más completo el proceso de toma de decisiones.

Este hecho hace que SimuRoute sea una buena opción para la búsqueda de rutas con criterios medioambientales, ya que lo que interesa en este proyecto es conseguir múltiples rutas con distintas características y que cumplan una serie de requisitos impuestos por el usuario. Muchas de las técnicas utilizadas normalmente para resolver problemas VRP proporcionan soluciones deterministas, de modo que siempre muestran la misma solución para un problema dado.

Esta es una de las principales razones por las cuales se ha decidido que el DSS se apoye en las rutas obtenidas por medio del SimuRoute. Una vez conseguidas las rutas candidatas mediante SimuRoute, se puede empezar a estudiar la preferencia de cada una en función de distintos criterios, para finalmente recomendar la solución más cercana al objetivo inicial, marcado por los criterios ambientales seleccionados por el usuario de la aplicación. En el caso de este Proyecto, las alternativas seleccionadas serán aquellas que satisfagan mejor los requisitos medioambientales impuestos, manteniendo un buen compromiso con el coste total de la ruta.

1.3.1. Funcionamiento de SimuRoute

El algoritmo SimuRoute, está basado en la combinación de la heurística Clarke & Wright (CWS) (*Clarke and Wright*, 1964), explicado anteriormente, con la simulación Monte Carlo (MCS). Además, el algoritmo incorpora algunas capacidades añadidas, algunas centradas en el uso de memoria, principalmente mediante el uso de una HashTable que guarda las mejore rutas conocidas, y otras basadas en técnicas de Splitting que utilizan las propiedades geométricas de las soluciones intermedias (*Ángel A. Juan, Javier Faulin et al*. 2009).

El uso combinado de la mejor gestión de la memoria junto con las técnicas de Splitting ha permitido significativas mejoras en el rendimiento dentro del grupo de los algoritmos CWS-MCS.

El funcionamiento general del algoritmo puede dividirse en 7 fases:

1. Una vez diseñado el problema VRP convenientemente, el algoritmo lo resuelve como si se tratara simplemente de un algoritmo C&W.
2. Se procesa el problema con el método C&W y se devuelve la solución. Contendrá el mejor conjunto de rutas posibles obtenidas mediante el C&W clásico.
3. Esta solución es guardada para poder compararla con las soluciones finales obtenidas por SimuRoute.
4. Se estudia la caracterización estadística de la solución C&W. Así tenemos unos valores de partida que el algoritmo toma como referencia para los cálculos posteriores.
5. El SimuRoute recoge de nuevo la entrada original del problema y, junto con la información obtenida gracias a la solución C&W, calcula las nuevas soluciones.
6. Una vez obtenidas las nuevas soluciones, son guardadas y reutilizadas para completar los procesos de caché y Splitting.
7. El algoritmo termina y devuelve la solución obtenida.

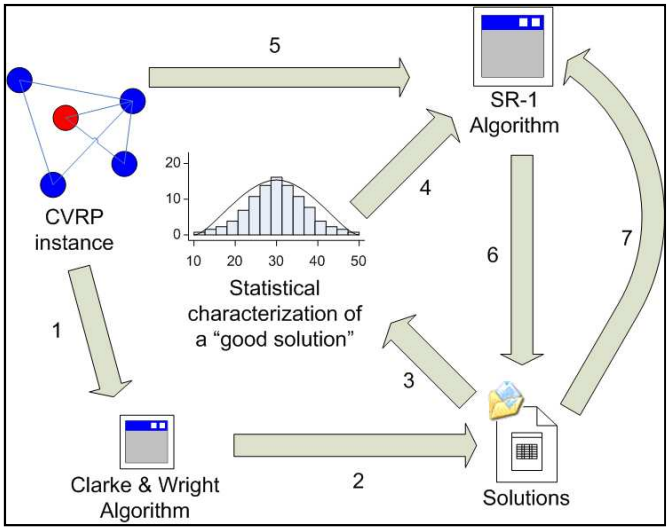


Ilustración 1.2. Esquema del funcionamiento general del SimuRoute (AA Juan, J Faulin, et Al. (2008))

A continuación se detallan las principales características que hacen de este algoritmo una buena opción para el cálculo de rutas del DSS.

Factor aleatorio

Una de las ideas principales de este método algorítmico es introducir un comportamiento aleatorio combinado con la heurística CWS, con el objetivo de mejorar el proceso de búsqueda en el espacio de soluciones posibles. Cada una de esas posibles soluciones consiste en una serie de rutas que parten desde el depósito y satisfacen las demandas de los nodos clientes visitándolos exactamente una vez a cada uno.

En cada paso del proceso de construcción de la solución, el algoritmo CWS siempre elige la arista que produce un mayor ahorro (Un comportamiento voraz o Greedy). En cambio, con el enfoque del SimuRoute, se le asigna a cada arista una probabilidad de ser elegida para su inclusión en la lista de ahorros. Esta probabilidad es coherente con el ahorro que produce cada arista, de modo que la que provoca el mayor ahorro tiene la mayor probabilidad de ser elegida, pero ya no es una elección segura. El resto de las aristas tienen una probabilidad asociada acorde con el ahorro que provocarían, pudiendo entonces ser elegidas en función de la probabilidad asignada.

Para conseguir este objetivo, se emplean distribuciones estadísticas geométricas durante el proceso de construcción de las soluciones CWS. Cada vez que se selecciona una arista de la lista de aristas disponibles, se emplea un valor **α** seleccionado aleatoriamente dentro de una distribución uniforme (a, b) donde $0 < a < b < 1$.

Con el parámetro **α,** se define la distribución geométrica específica que usará para asignar las probabilidades a cada arista elegible, de acuerdo con su posición dentro de la lista de ahorros.

De este modo se consigue que las aristas con mayores valores de ahorro sean siempre preferidas a la hora de ser elegidas de la lista. Pero al recalcularse en cada paso el valor de su probabilidad asociada, las posibilidades de elección de las aristas restantes cambian en cada iteración.

Utilización de una caché

Otra característica del SimuRoute es que utiliza un mecanismo de aprendizaje para conseguir llegar antes a la mejor solución. El mecanismo consiste en guardar en memoria, normalmente usando una Hash Table por mejorar los tiempos de acceso, para cada ruta generada, el mejor orden para recorrer sus nodos. Esta caché se actualiza siempre que se encuentre un orden mejor con un menor coste para recorrer un grupo de nodos determinado. De esta forma, vamos guardando en memoria las mejores soluciones para recorrer distintos subgrupos de nodos.

Al mismo tiempo, durante el proceso de construcción, las rutas contenidas en la caché son reutilizadas para mejorar las nuevas rutas generadas. Mientras la caché guarda las mejores soluciones para recorrer subconjuntos de nodos con un solo vehículo, las nuevas soluciones se beneficiarán de esta información durante su cálculo.

Gracias al uso de la caché, el algoritmo va acumulando experiencia y aprendiendo cuáles son los mejores caminos para recorrer distintos grupos de nodos, lo que al final repercute en mejores tiempos de cálculo y mayor calidad de las soluciones.

Política de Splitting

La otra idea importante en la que se fundamenta SimuRoute es la utilización de técnicas de Splitting o particionado. En algoritmia estás técnicas se denominan "*Divide y vencerás"* y se utilizan para dividir el problema en subproblemas del mismo tipo que el original, pero de menor tamaño, y por tanto, más fáciles de resolver.

El objetivo es encontrar un criterio para lograr dividir el conjunto de nodos original en distintos subconjuntos y después usar la metodología anteriormente descrita (combinando CWS y MCS junto con la técnica caché) para resolverlos independientemente.

El principio que se sigue para la división del problema es sencilla: Una vez tenemos una solución CWS mediante técnicas aleatorias, se usan criterios de proximidad o de geometría para dividir las rutas y sus correspondientes nodos.

A continuación, se aplica a cada subconjunto el proceso de construcción, con el objetivo de encontrar mejores soluciones con un menor coste.

El criterio de proximidad que utiliza el algoritmo está basado en la posición geométrica del centro de las rutas respecto al centro geométrico de la solución global, es decir, de todos los nodos. No se toma como referencia la posición del almacén, ya que existe multitud de problemas en los que el almacén no se encuentra centrado, sino que puede estar en alguna de los extremos. En el caso del ejemplo de navarra diseñado para este proyecto, el almacén se ha situado en Pamplona, por lo que prácticamente el centro de la solución final corresponde con el almacén.

En las siguientes imágenes se puede ver cómo funciona el proceso de Splitting:

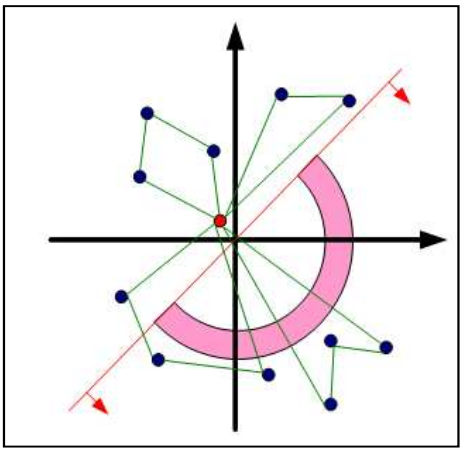


Ilustración 1.3. Partición del problema.

En este caso, se divide el problema original en dos siguiendo la diagonal trazada en rojo. Nos fijaremos en los nodos que han quedado por debajo de la misma.

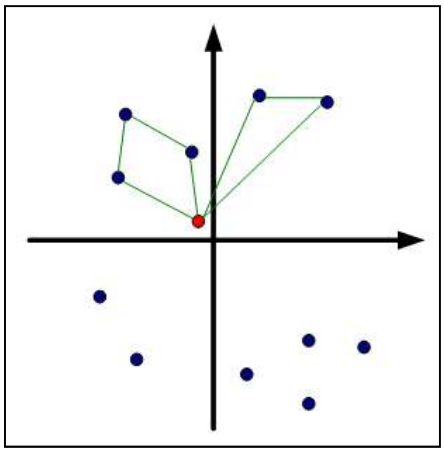


Ilustración 1.4. Creación de un subproblema menor.

Con los nodos que nos han quedado por debajo de la diagonal, se vuelve a tratar el problema con normalidad, es decir, el almacén sigue siendo el mismo pero los nodos cliente ahora sólo son los de abajo.

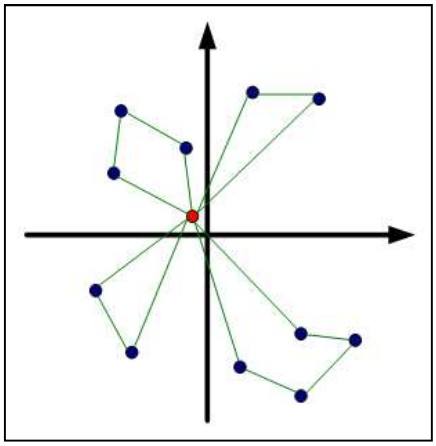


Ilustración 1.5. Nuevas rutas recalculadas. Unión con las demás rutas.

Ahora se reconstruye el problema con todos los nodos originales pero conservando las soluciones obtenidas en el paso anterior. De este modo pueden conseguirse soluciones mejores que las iniciales, ya que los subproblemas tratados son resueltos de formas más eficientes.

1.3.2. Pseudocódigo del SimuRoute

Proceso principal

El algoritmo recibe como entradas los nodos a ser servidos, además de un conjunto de restricciones, la matriz de costes y los parámetros de ejecución, incluyendo el generador de números aleatorios (rng), el número de mejores soluciones a guardar (nSol) y el número de iteraciones de primer y segundo nivel (nIter y nIterPerSplit).

Después, se calcula la matriz de ahorros y se construye una lista ordenada de ahorros. La lista resultante contiene las aristas que potencialmente pueden ser elegidas, ordenadas según sus ahorros asociados. Luego se obtiene una solución inicial aplicando el método CWS. Los costes asociados a esta solución CWS serán usados como límites superiores para los costes de la futura solución. De este modo nunca obtendremos una solución peor que el CWS.

En este momento empieza el proceso iterativo de primer nivel, donde se generan cientos o miles de nuevas soluciones intentando mejorar la solución CWS. En cada iteración de este nivel se construye una nueva solución usando el CWS aleatorio. Esa solución aleatorizada es analizada a continuación y procesada en el proceso de caché, donde se aprovechan las mejores soluciones guardadas de las iteraciones previas para mejorar la actual, cuando sea posible.

Si la solución actual mejora la CWS, es considerada como una solución prometedora y por tanto pasa a ser procesada con el proceso de Splitting.

En el proceso de Splitting se intenta mejorar la solución actual, primero considerando distintos subconjuntos de rutas y después aplicando un proceso iterativo de segundo nivel para cada uno de estos subconjuntos. Este proceso de segundo nivel hace uso del CWS aleatorizado y de los procesos de caché de nuevo.

Si la solución obtenida es buena, se guarda rápidamente en un array de manera ordenada.

```
procedure SR-GCWS-CS(vrpNodes, vrpConstraints, algParameters, costsMatrix)
// algParameters include rng, nSols, nIter and nIterPerSplit
1   savingsList = makeSavingsList(vrpNodes, costsMatrix);
2   cwsSol = constructCWSSol(vrpNodes, costsMatrix, savingsList, vrpConstraints);
3   while stopping criteria not satisfied do // it depends on nIter
4     vrpSol = constructRandomSol(vrpNodes, costsMatrix, savingsList,
          vrpConstraints, rng);
5     vrpSol = improveSolUsingRoutesCache(vrpSol, costsMatrix);
6     if vrpSol outperforms cwsSol then // vrpSol is a promising sol
7         vrpSol = improveSolUsingSplitting(vrpNodes, costsMatrix,
             savingsList, vrpConstraints, rng, nIterPerSplit, vrpSol, rCache);
8         bestSols = updateBestSolsList(vrpSol, bestSols, nSols);
9     fi
11  return bestSols;
12  end
```


Pseudocódigo 1.7. Proceso principal del SimuRoute. (SR-GCWS-CS 2009)

Proceso de que hace al CWS aleatorio

Este proceso parte de la solución inicial calculada por el método CWS clásico. A esta solución le aplica la selección de aristas pero con un componente de aleatoriedad. Como se ha explicado previamente, la asignación de probabilidades aleatorias está basada en el uso de distribuciones geométricas.

```
procedure constructRandomSol(nodes, cMatrix, sList, constraints, rng)
1   effList = copyList(sList);
2   sol = constructInitialSol(nodes, cMatrix); // sol = {(0,i,0) / i in nodes}
3   while effList contains edges do
4      e = selectEdgeAtRandom(effList, rng);
5      iNode = getOrigin(e);
6      jNode = getEnd(e);
7      iR = getRoute(iNode, sol);
8      jR = getRoute(jNode, sol);
9      if all CWS route-merging conditions are satisfied (see constraints) then
10        sol = mergeRoutesUsingEdge(e, iR, jR, sol); // see CWS heuristic
11     fi
12     deleteEdgeFromList(e, effList);
13  elihw
14  return sol;
end
```

Pseudocódigo 1.8. Proceso que construye soluciones CWS aleatorizadas. (SR-GCWS-CS 2009)

Proceso de selección aleatoria de aristas

El en siguiente método, recibe como entradas sólo dos atributos. Una lista que contiene las aristas elegibles y la variable “rng”, que contiene el factor de aleatoriedad a aplicar.

A partir de este valor se generan las probabilidades que después son asignadas a las aristas siguiendo una distribución geométrica.

Finalmente se devuelve una de las aristas, la cual ha sido seleccionada de forma aleatoria según su probabilidad asignada.

```
procedure selectEdgeAtRandom(list, rng)
1   beta = generateRandomNumber(rng, a, b); // e.g.: a = 0.06 and b = 0.22
2   randomValue = generateRandomNumber(rng, 0, 1);
3   n = 0;
4   cumulativeProbability = 0.0;
5   for each edge e in sorted list
6      eProbability = beta * (1 – beta)^n; // use geometric distribution
7      cumulativeProbability += eProbability;
8      if randomValue < cumulativeProbability then
9         return e;
10     else
11        n = n + 1;
12     fi
13  rof
14  k = generateIntegerRandomNumber(rng, 0, listSize); // k in [0, listSize)
15  return getEdgeAtPosition(k);
End
```

Pseudocódigo 1.9. Selección aleatoria de aristas. (SR-GCWS-CS 2009)

Mecanismo de aprendizaje a través de la caché de rutas

A continuación se muestra el código que representa el comportamiento del proceso de la caché de rutas, el cual normalmente supone una mejora rápida de las soluciones. En este caso, la Hash Table utilizada siempre guarda el mejor orden conocido para visitar un subconjunto de nodos cliente en una ruta.

Como se ha explicado anteriormente, esto puede ser considerado como un mecanismo de aprendizaje sencillo, que hace que el algoritmo sea más eficiente cuantas más soluciones se generan.

Como entradas tomará la mejor solución conocida, la matriz de costes actual y la caché existente en ese momento.

Finalmente devuelve una nueva solución actualizada, siempre y cuando sea mejor que la anterior solución encontrada.

Por razones de eficiencia, se utiliza una Hash table en lugar de un Array o cualquier otra estructura de datos, ya que permite realizar entradas, búsquedas y actualizaciones de una forma rápida y cómoda.

```
procedure improveSolUsingRoutesCache(sol, cMatrix, rCache)
1   table = makeHashTable();
2   for each route r in sol
3      hashCode = getRouteHashCode(r);
4      if hashCode already in table then
5         routeInTable = getRouteInHashTable(rCache, hashCode);
6         if routeInTable outperforms r then
7            r = routeInTable;
8            sol = updateRouteInSol(r, sol);
9         else
10           r = improveNodesOrder(r, cMatrix); // aims to avoid knots in r
11           rCache = updateRouteInHashTable(rCache, hashCode, r);
12        fi
13     else
14        r = improveNodesOrder(r, cMatrix);
15        rCache = putRouteInHashTable(rCache, hashCode, r);
16     fi
17  rof
18  return sol;
End
```

Pseudocódigo 1.10. Uso de la caché de rutas. (SR-GCWS-CS 2009)

Aplicación de Splitting para reducir la dimensión del problema

En el siguiente proceso se aplica la idea de dividir el problema en varios subproblemas más pequeños para su resolución por separado.

En este caso es importante definir bien el criterio por el cual se va a dividir el problema. Un criterio de aproximación demasiado estricto puede dar lugar a situaciones en las que no podamos encontrar subproblemas. Al contrario, un criterio poco exigente puede dar como resultado una gran cantidad de subproblemas distintos, haciendo que los cálculos crezcan exponencialmente, perjudicando el tiempo de cómputo.

```
procedure improveSolUsingSplitting(vrpNodes, costsMatrix,savingsList,vrpConstraints,
rng, nIterPerSplit, vrpSol, rCache)
1   vrpCenter = calcGeometricCenter(vrpNodes); // (x-bar, y-bar) of nodes
2   sol = vrpSol; // initial sol
3   while a new splitting policy is available then
4      allRoutes = getRoutes(sol);
5      routesCenters = calcRoutesGeometricCenters(allRoutes);
6      frontRoutes = selectFrontRoutes(allRoutes, routesCenters, policy);
7      backRoutes = substractRoutesFromList(frontRoutes, allRoutes);
8      frontSubSol = makeSubSol(frontRoutes);
9      backSubSol = makeSubSol(backRoutes);
10     frontNodes = getNodes(frontSubSol);
11     splitSavingsList = makeSavingsList(frontNodes, savingsMatrix);
12     splitSavingsList = sortList(splitSavingsList);
13     while stopping criteria not satisfied (1<= iter <= nIterPerSplit) then
14        newSubSol = constructRandomSol(frontNodes, costsMatrix,
             splitSavingsList, constraints, rng);
15        newSubSol = improveSolUsingRoutesCache(newSol, costsMatrix, rCache);
16        if newSubSol outperforms frontSubSol then
17           frontSubSol = newSubSol;
18        fi
19     elihw
20     newSol = unifySubSols(frontSubSol, backSubSol);
21     if newSol outperforms sol then
22        sol = newSol;
23     fi
24  elihw
25  return sol;
end
```

Pseudocódigo 1.11. Proceso de Splitting. (SR-GCWS-CS 2009)

2. SISTEMAS DE INFORMACIÓN Y DSS

Hoy en día, y desde hace décadas, las herramientas informáticas se han hecho imprescindibles para la resolución de todo tipo de problemas, desde los más cotidianos o relacionados con el ocio, hasta los más complejos retos científicos. Están presentes en el mundo empresarial, haciendo que operaciones y transacciones que en el pasado costaban días, ahora sean cuestión de segundos. Además se han implantado en la sociedad nuevas costumbres, como el comercio electrónico o el e-mail, que están sustituyendo las formas tradicionales de comercio y de comunicación interpersonal.

En éste capítulo se va a explicar qué es un DSS, atendiendo a cómo han evolucionado los sistemas de información en los últimos años. Los avances en éste ámbito de la informática, normalmente, han sido resultado de los esfuerzos del mundo empresarial por conseguir herramientas software cada vez más completas y potentes. La fuerte competencia hace que las empresas destinen recursos a la adquisición y mejora de aplicaciones informáticas, las cuales facilitan y optimizan los diferentes procesos de la empresa, como puede ser el de la toma de decisiones.

Desde mediados del siglo XX, la evolución de la informática ha sido un proceso constante e incansable, que se ha dado de forma paralela en sus dos vertientes, hardware y software.

Por parte del hardware, principalmente han sido el aumento de la potencia computacional de los procesadores, así como el incremento del volumen y velocidad de las memorias, los que han permitido plantear y resolver problemas cuyo volumen de datos y complejidad, hacían que en el pasado fueran impracticables.

Del mismo modo, los programas y aplicaciones informáticas se han ido rediseñando y evolucionando a la par que el hardware. Los sistemas operativos actuales permiten el correcto funcionamiento de los programas en máquinas que permiten la multiprogramación y la ejecución de hilos en paralelo. Este tipo de avances han permitido que se desarrollen nuevas técnicas algorítmicas avanzadas, que posibilitan encontrar soluciones aceptables para muchos problemas irresolubles de forma directa.

Las redes de comunicaciones han contribuido de manera notoria a la mejora de las herramientas de cálculo. Existen numerosos proyectos que se sirven de la multiprogramación y de los sistemas distribuidos para aumentar enormemente la potencia de cálculo acumulada.

Gracias a esta combinación de avances en hardware, software y redes de las últimas décadas, hemos podido diseñar programas informáticos mucho más complejos, con capacidades muy especializadas, para resolver infinidad de diferentes tipos de problemas.

2.1. Sistemas de información

Un sistema de información es un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades de una empresa o negocio. Cualquier sistema de información requiere dos factores para su funcionamiento:

- El equipo computacional: el hardware necesario para que el sistema de información pueda operar.
- El recurso humano que interactúa con el Sistema de Información, el cual está formado por las personas que utilizan el sistema.

Un sistema de información realiza cuatro actividades básicas:

1. **Entrada de Información:** Es el proceso mediante el cual el Sistema de Información toma los datos que requiere para procesar la información. Las entradas pueden ser manuales o automáticas. Las manuales son aquellas que se proporcionan en forma directa por el usuario, mientras que las automáticas son datos o información que provienen o son tomados de otros sistemas o módulos.

   Las unidades típicas de entrada de datos a las computadoras son las terminales, las cintas magnéticas, las unidades de diskette, los códigos de barras, los escáneres, la voz, los monitores sensibles al tacto, el teclado y el mouse, entre otras.

2. **Almacenamiento de información:** El almacenamiento es una de las actividades o capacidades más importantes que tiene una computadora, ya que a través de esta propiedad el sistema puede recordar la información guardada en la sección o proceso anterior. Esta información suele ser almacenada en estructuras de información denominadas archivos. La unidad típica de almacenamiento son los discos magnéticos o discos duros y los discos compactos (CD-ROM).

3. **Procesamiento de Información:** Es la capacidad del Sistema de Información para efectuar cálculos de acuerdo con una secuencia de operaciones preestablecida. Estos cálculos pueden efectuarse con datos introducidos recientemente en el sistema o bien con datos que están almacenados. Esta característica de los sistemas permite la transformación de datos fuente en información que puede ser utilizada para la toma de decisiones, lo que hace posible, entre otras cosas, que un tomador de decisiones genere una proyección financiera a partir de los datos que contiene un estado de resultados o un balance general de un año base.

4. **Salida de Información:** La salida es la capacidad de un Sistema de Información para sacar la información procesada o bien datos de entrada al exterior. Las unidades típicas de salida son las impresoras, terminales, monitores,…entre otros. Es importante aclarar que la salida de un Sistema de Información puede constituir la entrada a otro Sistema de Información o módulo.

2.1.1. Tipos y Usos de los Sistemas de Información

Durante los próximos años, los Sistemas de Información cumplirán tres objetivos básicos dentro de las organizaciones:

1. Automatización de procesos operativos.
2. Proporcionar información que sirva de apoyo al proceso de toma de decisiones.
3. Lograr ventajas competitivas a través de su implantación y uso.

Los Sistemas de Información que logran la automatización de procesos operativos dentro de una organización, son llamados frecuentemente Sistemas Transaccionales, ya que su función primordial consiste en procesar transacciones tales como pagos, cobros, pólizas, entradas, salidas, etc.

Por otra parte, los Sistemas de Información que apoyan el proceso de toma de decisiones son los Sistemas de Soporte a la Toma de Decisiones, Sistemas para la Toma de Decisión de Grupo, Sistemas Expertos de Soporte a la Toma de Decisiones y Sistema de Información para Ejecutivos.

El tercer tipo de sistema, de acuerdo con su uso u objetivos que cumplen, es el de los Sistemas Estratégicos, los cuales se desarrollan en las organizaciones con el fin de lograr ventajas competitivas, a través del uso de la tecnología de información.

A continuación se mencionan las principales características de estos tipos de Sistemas de Información.

Sistemas Transaccionales. Sus principales características son:

- A través de éstos suelen lograrse ahorros significativos de mano de obra, debido a que automatizan tareas operativas de la organización.

- Con frecuencia son el primer tipo de Sistemas de Información que se implanta en las organizaciones. Se empieza apoyando las tareas a nivel operativo de la organización.

- Son intensivos en entrada y salid de información; sus cálculos y procesos suelen ser simples y poco sofisticados.

- Tienen la propiedad de ser recolectores de información, es decir, a través de estos sistemas se cargan las grandes bases de información para su explotación posterior.

- Son fáciles de justificar ante la dirección general, ya que sus beneficios son visibles y palpables.

Sistemas de Apoyo de las Decisiones. Las principales características de estos son:

- Suelen introducirse después de haber implantado los Sistemas Transaccionales más relevantes de la empresa, ya que estos últimos constituyen su plataforma de información.

- La información que generan sirve de apoyo a los mandos intermedios y a la alta administración en el proceso de toma de decisiones.

- Suelen ser intensivos en cálculos y escasos en entradas y salidas de información. Así, por ejemplo, un modelo de planificación financiera requiere poca información de entrada, genera poca información como resultado, pero puede realizar muchos cálculos durante su proceso.

- No suelen ahorrar mano de obra. Debido a ello, la justificación económica para el desarrollo de estos sistemas es difícil, ya que no se conocen los ingresos del proyecto de inversión.

- Suelen ser Sistemas de Información interactivos y amigables, con altos estándares de diseño gráfico y visual, ya que están dirigidos al usuario final.

- Apoyan la toma de decisiones que, por su misma naturaleza son repetitivos y de decisiones no estructuradas que no suelen repetirse.

- Estos sistemas pueden ser desarrollados directamente por el usuario final sin la participación operativa de los analistas y programadores del área de informática.

Este tipo de sistemas puede incluir la programación de la producción, compra de materiales, flujo de fondos, proyecciones financieras, modelos de simulación de negocios, modelos de inventarios, etc.

Sistemas Estratégicos. Sus principales características son:

- Su función primordial no es apoyar la automatización de procesos operativos ni proporcionar información para apoyar la toma de decisiones.

- Suelen desarrollarse dentro de la organización, por lo tanto no pueden adaptarse fácilmente a paquetes disponibles en el mercado.

- Típicamente su forma de desarrollo es a base de incrementos y a través de su evolución dentro de la organización. Se inicia con un proceso o función en particular y a partir de ahí se van agregando nuevas funciones o procesos.

- Su función es lograr ventajas que los competidores no posean, tales como ventajas en costos y servicios diferenciados con clientes y proveedores. En este contexto, los Sistema Estratégicos son creadores de barreras de entrada al negocio

- Apoyan el proceso de innovación de productos y proceso dentro de la empresa debido a que buscan ventajas respecto a los competidores y una forma de hacerlo en innovando o creando productos y procesos.

Un ejemplo de estos Sistemas de Información dentro de la empresa puede ser un sistema MRP (Manufacturing Resoure Planning) enfocado a reducir sustancialmente el desperdicio en el proceso productivo, o bien, un Centro de Información que proporcione todo tipo de información; como situación de créditos, embarques, tiempos de entrega, etc. En este contexto los ejemplos anteriores constituyen un Sistema de Información Estratégico si y sólo sí, apoyan o dan forma a la estructura competitiva de la empresa.

Por último, es importante aclarar que algunos autores consideran un cuarto tipo de sistemas de información denominado Sistemas Personales de Información, el cual está enfocado a incrementar la productividad de sus usuarios.

2.1.2. Implantación de los Sistemas de Información

De la sección anterior se desprende la evolución que tienen los Sistemas de Información en las organizaciones. Con frecuencia se implantan en forma inicial los Sistemas Transaccionales y, posteriormente, se introducen los Sistemas de Apoyo a las Decisiones. Por último, se desarrollan los Sistemas Estratégicos que dan forma a la estructura competitiva de la empresa.

En la década de los setenta, se desarrolló una teoría que impactó el proceso de planificación de los recursos y las actividades de la informática (*Nolan R.* 1973).

Según Nolan, la función de la Informática en las organizaciones evoluciona a través de ciertas etapas de crecimiento, las cuales se explican a continuación:

- Comienza con la adquisición de la primera computadora y normalmente se justifica por el ahorro de mano de obra y el exceso de papeles.
- Las aplicaciones típicas que se implantan son los Sistemas Transaccionales tales como nóminas o contabilidad.

- El pequeño Departamento de Sistemas depende en la mayoría de los casos del área de contabilidad.

- El tipo de administración empleada es escaso y la función de los sistemas suele ser manejada por un administrador que no posee una preparación formal en el área de computación.

- El personal que labora en este pequeño departamento consta a lo sumo de un operador y/o un programador. Este último podrá estar bajo el régimen de honorarios, o bien, puede recibirse el soporte de algún fabricante local de programas de aplicación.

- En esta etapa es importante estar consciente de la resistencia al cambio del personal y usuario que están involucrados en los primeros sistemas que se desarrollan, ya que estos sistemas son importantes en el ahorro de mano de obra.

- Esta etapa termina con la implantación exitosa del primer Sistema de Información. Cabe recalcar que algunas organizaciones pueden vivir varias etapas de inicio en las que la resistencia al cambio por parte de los primeros usuarios involucrados aborta el intento de introducir la computadora a la empresa.

Etapa de contagio o expansión. Los aspectos sobresalientes que permiten diagnosticar rápido que una empresa se encuentra en esta etapa son:

- Se inicia con la implantación exitosa del primer Sistema de Información en la organización. Como consecuencia de lo anterior, el primer ejecutivo usuario se transforma en el paradigma o persona que se habrá que imitar.

- Las aplicaciones que con frecuencia se implantan en esta etapa son el resto de los Sistemas Transaccionales no desarrollados en la etapa de inicio, tales como facturación, inventarios, control de pedidos de clientes y proveedores, cheques, etc.

- El pequeño departamento es promovido a una categoría superior, donde depende de la Gerencia Administrativa o Contraloría.

- El tipo de administración empleado está orientado hacia la venta de aplicaciones a todos los usuarios de la organización; en este punto suele contratarse a un especialista de la función con preparación académica en el área de sistemas.

- Se inicia la contratación de personal especializado y nacen puestos tales como analista de sistemas, analista-programador, programador de sistemas, jefe de desarrollo, jefe de soporte técnico, etc.

- Las aplicaciones desarrolladas carecen de interfaces automáticas entre ellas, de tal forma que las salidas que produce un sistema se tienen que alimentar en forma manual a otro sistema, con la consecuente irritación de los usuarios.

- Los gastos por concepto de sistemas empiezan a crecer en forma importante, lo que marca la pauta para iniciar la racionalización en el uso de los recursos computacionales dentro de la empresa. Este problema y el inicio de su solución marcan el paso a la siguiente etapa.

Etapa de control o formalización. Para identificar a una empresa que transita por esta etapa es necesario considerar los siguientes elementos:

- Esta etapa de evolución de la Informática dentro de las empresas se inicia con la necesidad de controlar el uso de los recursos computacionales a través de las técnicas presupuestarias de base cero (partiendo de que no se tienen nada) y la implantación de sistemas de cargos a usuarios (por el servicio que se presta).

- Las aplicaciones están orientadas a facilitar el control de las operaciones del negocio para hacerlas más eficaces, tales como sistemas para control de flujo de fondos, control

de órdenes de compra a proveedores, control de inventarios, control y manejo de proyectos, etc.

- El departamento de sistemas de la empresa suele ubicarse en una posición gerencial, dependiendo del organigrama de la Dirección de Administración o Finanzas.

- El tipo de administración empleado dentro del área de Informática se orienta al control administrativo y a la justificación económica de las aplicaciones a desarrollar. Nace la necesidad de establecer criterios para las prioridades en el desarrollo de nuevas aplicaciones. La cartera de aplicaciones pendientes por desarrollar empieza a crecer.

- En esta etapa se inician el desarrollo y la implantación de estándares de trabajo dentro del departamento, tales como: estándares de documentación, control de proyectos, desarrollo y diseño de sistemas, auditoría de sistemas y programación.

- Se integra a la organización del departamento de sistemas a personal con habilidades administrativas y preparado técnicamente.

- Se inicia el desarrollo de interfaces automáticas entre los diferentes sistemas.

Etapa de integración. Las características de esta etapa son las siguientes:

- La integración de los datos y de los sistemas surge como un resultado directo de la centralización del departamento de sistemas bajo una sola estructura administrativa.

- Las nuevas tecnologías relacionadas con base de datos, sistemas administradores de bases de datos y lenguajes de cuarta generación, hicieron posible la integración.

- En esta etapa surge la primera hoja electrónica de cálculo comercial y los usuarios inician haciendo sus propias aplicaciones. Esta herramienta ayudó mucho a que los usuarios hicieran su propio trabajo y no tuvieran que esperar a que sus propuestas de sistemas fueran cumplidas.

- El costo del equipo y del software disminuyó por lo cual estuvo al alcance de más usuarios.

- En forma paralela a los cambios tecnológicos, cambió el rol del usuario y del departamento de Sistemas de Información. El departamento de sistemas evolucionó hacia una estructura descentralizada, permitiendo al usuario utilizar herramientas para el desarrollo de sistemas.

- Los usuarios y el departamento de sistema iniciaron el desarrollo de nuevos sistemas, reemplazando los sistemas antiguos, en beneficio de la organización.

Etapa de administración de datos. Entre las características que destacan en esta etapa están las siguientes:

- El departamento de Sistemas de Información reconoce que la información es un recurso muy valioso que debe estar accesible para todos los usuarios.

- Para poder cumplir con lo anterior resulta necesario administrar los datos en forma apropiada, es decir, almacenarlos y mantenerlos en forma adecuada para que los usuarios puedan utilizar y compartir este recurso.

- El usuario de la información adquiere la responsabilidad de la integridad de la misma y debe manejar niveles de acceso diferentes.

Etapa de madurez. Entre los aspectos sobresalientes que indican que una empresa se encuentra en esta etapa, se incluyen los siguientes:

- Al llegar a esta etapa, la Informática dentro de la organización se encuentra definida como una función básica y se ubica en los primeros niveles del organigrama (dirección).

- Los sistemas que se desarrollan son Sistemas de Manufactura Integrados por Computadora, Sistemas Basados en el Conocimiento y Sistemas Expertos, Sistemas de Soporte a las Decisiones, Sistemas Estratégicos y, en general, aplicaciones que proporcionan información para las decisiones de alta administración y aplicaciones de carácter estratégico.

- En esta etapa se tienen las aplicaciones desarrolladas en la tecnología de base de datos y se logra la integración de redes de comunicaciones con terminales en lugares remotos, a través del uso de recursos computacionales.

2.2. Proceso de toma de decisiones

Un problema que surge continuamente en cualquier ámbito de la vida, es el de la toma de decisiones. Concretamente, en el mundo empresarial se deben tomar decisiones constantemente, eligiendo una opción y desechando muchas otras.

Si la decisión tomada no es adecuada, el coste para la empresa puede ser importante. No sólo es el coste de aplicar una mala decisión, además hay que sumar el coste de no haber elegido la opción correcta, perdiendo los posibles beneficios que ésta podría haber aportado. Dependiendo del nivel de la decisión y de otros muchos factores, como el estado de la economía interna de la empresa, o el valor estratégico de la decisión, un resultado negativo puede llegar a ser catastrófico.

Podemos decir que es el proceso durante el cual la persona debe escoger entre dos o más alternativas. (*F. Burstein, C.W. Holsapple* 2008) Para los administradores de cualquier empresa, el proceso de toma de decisión es sin duda una de las mayores responsabilidades.

La toma de decisiones en una organización se circunscribe a una serie de personas que están apoyando el mismo proyecto. Debemos empezar por hacer una selección de decisiones, y esta selección es una de las tareas de gran trascendencia.

Con frecuencia se dice que las decisiones son algo así como el motor de los negocios y en efecto, de la adecuada selección de alternativas depende en gran parte el éxito de cualquier organización.

Los administradores consideran a veces la toma de decisiones como su trabajo principal, porque constantemente tienen que decidir lo que debe hacerse, quién ha de hacerlo, cuándo y dónde, y en ocasiones hasta cómo se hará.

La toma de decisiones en una organización invade cuatro funciones administrativas que son: planificación, organización, dirección y control.

1. **Planificación**: Selección de misiones y objetivos así como de las acciones para cumplirlas. Es aquí donde se da verdaderamente la toma de decisiones. Esto implica responder a las siguientes cuestiones:

   ¿Cuáles son los objetivos de la organización, a largo plazo?

   ¿Qué estrategias son mejores para lograr este objetivo?

   ¿Cuáles deben ser los objetivos a corto plazo?

   ¿Cuán altas deben ser las metas individuales?

2. **Organización:** Establecimiento de la estructura que desempeñan los individuos dentro de la organización.

   ¿Cuánta centralización debe existir en la organización?

   ¿Cómo deben diseñarse los puestos?

   ¿Quién está mejor calificado para ocupar un puesto vacante?

¿Cuándo debe una organización instrumentar una estructura diferente?

3. **Dirección:** Esta función requiere que los administradores influyan en los individuos para el cumplimiento de las metas organizacionales y grupales.

   ¿Cómo manejo a un grupo de trabajadores que parecen tener una motivación baja?

   ¿Cuál es el estilo de liderazgo más eficaz para una situación dada?

   ¿Cómo afectará un cambio específico a la productividad del trabajador?

   ¿Cuándo es adecuado estimular el conflicto?

4. **Control:** Es la medición y corrección del desempeño individual y organizacional de manera tal que se puedan lograr los planes.

   ¿Qué actividades en la organización necesitan ser controladas?

   ¿Cómo deben controlarse estas actividades?

   ¿Cuándo es significativa una desviación en el desempeño?

   ¿Cuándo la organización está desempeñándose de manera efectiva?

2.2.1. Proceso racional de toma de decisiones

Análisis que requiere de una meta y una comprensión clara de las alternativas mediante las que se puede alcanzar una meta, un análisis y evaluación de las alternativas en término de la meta deseada, la información necesaria y el deseo de optimizar.

Cuando un administrador se enfrenta a una toma de decisión, además de comprender la situación que se presenta, debe tener la capacidad de analizar, evaluar, reunir alternativas, considerar las variables, es decir, aplicar estas técnicas para encontrar soluciones razonables; podemos decir entonces, que se trata de una toma de decisión basada en la racionalidad.

Aunque muchas decisiones administrativas se toman con el deseo de salir adelante en una forma tan segura como sea posible, la mayoría de los administradores intentan tomar las mejores decisiones que puedan, dentro de los límites de la racionalidad y de acuerdo con el tamaño y la naturaleza de los riesgos implícitos.

De los procesos existentes para la toma de decisiones, este es catalogado como "el proceso ideal". En su desarrollo, el administrador debe:

- Determinar la necesidad de una decisión.

  El proceso de toma de decisiones comienza con el reconocimiento de que se necesita tomar una decisión. Ese reconocimiento lo genera la existencia de un problema o una disparidad entre cierto estado deseado y la condición real del momento.

- Identificar los criterios de decisión.

  Una vez determinada la necesidad de tomar una decisión, se deben identificar los criterios que sean importantes para la misma.

- Asignar peso a los criterios.

  Los criterios enumerados en el paso previo no tienen igual importancia. Es necesario ponderar cada uno de ellos y priorizar su importancia en la decisión.Desarrollar todas las alternativas.

- Desplegar las alternativas.

  La persona que debe tomar una decisión tiene que elaborar una lista de todas las alternativas disponibles para la solución de un determinado problema.

- Evaluar las alternativas.

  La evaluación de cada alternativa se hace analizándola con respecto al criterio ponderado.

  Una vez identificadas las alternativas, el tomador de decisiones tiene que evaluar de manera crítica cada una de ellas. Las ventajas y desventajas de cada alternativa resultan evidentes cuando son comparadas.

- Seleccionar la mejor alternativa.

  Una vez seleccionada la mejor alternativa se llegó al final del proceso de toma de decisiones. En el proceso racional, esta selección es bastante simple. El tomador de decisiones sólo tiene que escoger la alternativa que tuvo la calificación más alta en el paso número cinco.

El tomador de decisiones debe ser totalmente objetivo y lógico a la hora de tomarlas. Tiene que tener una meta clara y todas las acciones en el proceso de toma de decisiones llevan de manera consistente a la selección de aquella alternativa que maximizará la meta.

Vamos a analizar la toma de decisiones de una forma totalmente racional:

- Orientada a un objetivo. Cuando se deben tomar decisiones, no deben existir conflictos acerca del objetivo final. El lograr los fines es lo que motiva que tengamos que decidir la solución que más se ajusta a las necesidades concretas.
- Todas las opciones son conocidas. El tomador de decisiones tiene que conocer las posibles consecuencias de su determinación. Así mismo tiene claros todos los criterios y puede enumerar todas las alternativas posibles.
- Las preferencias son claras. Se supone que se pueden asignar valores numéricos y establecer un orden de preferencia para todos los criterios y alternativas posibles.

El proceso creativo

El proceso creativo no suele ser simple ni lineal. Por lo general se compone, de cuatro fases sobrepuestas que interactúan entre sí:

- La primera fase, exploración inconsciente, es difícil de explicar en razón de que ocurre fuera de los límites de la conciencia. Usualmente implica la abstracción de un problema, cuya determinación mental es probable que sea muy vaga. Sin embargo, los

administradores que trabajan bajo intensas presiones de tiempo suelen tomar decisiones prematuras antes que ocuparse detenidamente de problemas ambiguos y escasamente definidos.

- En la segunda fase, la intuición sirve de enlace entre el inconsciente y la conciencia. Esta etapa puede implicar una combinación de factores aparentemente contradictorios a primera vista.

  La intuición precisa de tiempo para funcionar. Supone para los individuos la detección de nuevas combinaciones y la integración de conceptos e ideas diversos. Para ello es necesario profundizar en el análisis de un problema. El pensamiento intuitivo puede inducirse mediante técnicas como la lluvia de ideas.

- El discernimiento, tercera fase del proceso creativo, es resultado sobre todo del trabajo intenso. Para desarrollar un producto útil, un nuevo servicio o un nuevo proceso, por ejemplo, son necesarias muchas ideas.

- La última fase del proceso creativo es la formulación o verificación lógica. El discernimiento debe someterse a la prueba de la lógica o de la experimentación. Esto se logra mediante la persistente reflexión en una idea o pidiendo críticas a los demás.

2.2.2. Etapas De La Toma De Decisiones

Identificación y diagnóstico del problema:

Reconocemos en la fase inicial el problema que deseamos solucionar, teniendo en cuenta el estado actual con respecto al estado deseado. Una vez que el problema es identificado se debe realizar el diagnóstico y luego de esto podremos desarrollar las medidas correctivas.

Generación de soluciones alternativas:

La solución de los problemas puede lograrse por varios caminos y no sólo seleccionar entre dos alternativas, se pueden formular hipótesis ya que con la alternativa hay incertidumbres.

Evaluación de alternativas:

La tercera etapa implica la determinación del valor o la adecuación de las alternativas que se generaron. ¿Cuál solución será la mejor?
Los gerentes deben considerar distintos tipos de consecuencia. Por supuesto que deben intentar predecir los efectos sobre las medidas financieras u otras medidas de desarrollo. Pero también existen otras consecuencias menos definidas que hay que atender. Las decisiones establecen un precedente y hay que determinar si este será una ayuda o un obstáculo en el futuro.

Por supuesto, no es posible predecir los resultados con toda precisión. Entonces pueden generar planes de contingencia, esto es, curso alternativo de acción que se pueden implantar con base en el desarrollo de los acontecimientos.

Selección de la mejor alternativa:

Cuando el administrador ha considerado las posibles consecuencias de sus opciones, ya está en condiciones de tomar la decisión. Debe considerar tres términos muy importantes. Estos son: maximizar, satisfacer y optimizar.

- Maximizar: es tomar la mejor decisión posible
- Satisfacer: es la elección de la primera opción que sea mínimamente aceptable o adecuada, y de esta forma se satisface una meta o criterio buscado.
- Optimizar: Es el mejor equilibrio posible entre distintas metas.

Implementación de la decisión:

El proceso no finaliza cuando la decisión se toma; esta debe ser implementada. Bien puede ser que quienes participen en la elección de una decisión sean quienes procedan a implementarla, como en otras ocasiones delegan dicha responsabilidad en otras personas. Debe existir la comprensión total sobre la elección de la toma de decisión en sí, las razones que la motivan y sobre todo debe existir el compromiso de su implementación exitosa. Para tal fin, las personas que participan en esta fase del proceso, deberían estar involucradas desde las primeras etapas que anteriormente hemos mencionado.

Evaluación de la decisión:

Forma parte de la etapa final de este proceso. Se recopila toda la información que nos indique la forma como funciona una decisión, es decir, es un proceso de retroalimentación que podría ser positiva o negativa.

Si la retroalimentación es positiva, pues entonces nos indica que podemos continuar sin problemas y que incluso se podría aplicar la misma decisión a otras áreas de la organización.

Si por el contrario, la retroalimentación es negativa, podría ser que:

1) tal vez la implementación requiera de más tiempo, recursos, esfuerzos o pensamiento o

2) nos puede indicar que la decisión fue equivocada, para lo cual debemos volver al principio del proceso redefinición del problema. Si esto ocurriera, sin duda tendríamos más información y probablemente sugerencias que nos ayudarían a evitar los errores cometidos en el primer intento.

2.2.3. Cualidades personales para la toma de decisiones

Sin lugar a dudas existen ciertas cualidades que hacen que los tomadores de decisión sean buenos o malos.

Existen cuatro cualidades que tienen mayor importancia a la hora de analizar al tomador de decisiones: experiencia, buen juicio, creatividad y habilidades cuantitativas. Otras cualidades podrán ser relevantes, pero estas cuatro forman los requisitos fundamentales.

- Experiencia: Es lógico suponer que la habilidad de un mando para tomar decisiones crece con la experiencia. Para situaciones mal estructuradas o nuevas, la experiencia puede acarrear ventajas y desventajas. La principal desventaja es que las lecciones de experiencia puedan ser inadecuadas por completo para el nuevo problema, resultando una decisión errónea. Pero también puede ser una gran ventaja, pues da elementos para diferenciar entre situaciones bien o mal estructuradas.
- Buen juicio: Se utiliza el término juicio para referirnos a la habilidad de evaluar información de forma inteligente. Está constituido por el sentido común, la madurez, la habilidad de razonamiento y la experiencia del tomador de decisiones. Por lo tanto se supone que el juicio mejora con la edad y la experiencia.

  El buen juicio se demuestra a través de ciertas habilidades para percibir información importante, sopesar su importancia y evaluarla. El juicio es más valioso en el manejo de problemas mal estructurados o nuevos, porque precisamente de ese juicio el tomador de decisiones sacará determinaciones y aplicará criterios para entender el problema y simplificarlo, sin distorsionarlo con la realidad.

  Un juicio se desarrolla de la siguiente manera: basado en la información disponible y en su propia experiencia anterior, el tomador de decisiones establece parámetros conformados por: los hechos, las opiniones y el conocimiento en general.
- Creatividad: La creatividad designa la habilidad del tomador de decisiones para combinar o asociar ideas de manera única, para lograr un resultado nuevo y útil.

  El tomador de decisiones creativo es capaz de captar y entender el problema de manera más amplia, aún de ver las consecuencias que otros pasan por alto. Sin embargo el mayor valor de la creatividad está en el desarrollo de alternativas. Son creativos y pueden generar suficientes ideas para encontrar el camino más corto y efectivo al problema.
- Habilidades cuantitativas: Esta es la habilidad de emplear técnicas presentadas como métodos cuantitativos o investigación de operaciones, como pueden ser: la programación lineal, teoría de líneas de espera y modelos de inventarios. Estas herramientas ayudan a los mandos a tomar decisiones efectivas. Pero es muy importante no olvidar que las habilidades cuantitativas no deben, ni pueden reemplazar al buen juicio en el proceso de toma de decisiones.

2.2.4. Importancia de la toma de decisiones

Es importante porque mediante el empleo de un buen juicio, la Toma de Decisiones nos indica que un problema o situación es valorado y considerado profundamente para elegir el mejor camino a seguir según las diferentes alternativas y operaciones.

En la Toma de Decisiones, considerar un problema y llegar a una conclusión válida, significa que se han examinado todas las alternativas y que la elección ha sido correcta. Dicho pensamiento lógico aumentará la confianza en la capacidad para juzgar y controlar situaciones. Uno de los enfoques más competitivos de investigación y análisis para la toma de las decisiones es la investigación operativa, puesto que esta es una herramienta importante para la administración de la producción y las operaciones.

La toma de decisiones, es considerada una parte importante del proceso de planificación cuando ya se conoce una oportunidad y una meta. El núcleo de la planificación es realmente el proceso de tomar la decisión, y se podría visualizar de la siguiente manera:

a. Elaboración de premisas y requisitos.
b. Identificación de alternativas.
c. Evaluación de alternativas teniendo en cuenta la meta deseada.
d. Elección de una alternativa, es decir, tomar una decisión.

2.3. DECISION SUPPORT SYSTEMS (DSS)

Un Sistema de Soporte a la Decisión (DSS por sus siglas en ingles) es, básicamente, un sistema de información que combina diferentes modelos y datos para intentar ofrecer la mejor solución para un problema concreto (*F. Burstein, C.W. Holsapple.* 2008).

Los DSS son una clase específica de sistemas de información basados en equipos de cómputo y que tienen como finalidad apoyar las actividades de toma de decisiones en la organización. Un DSS diseñado apropiadamente es un sistema de software interactivo orientado a ayudar al personal clave dentro de la organización -que son quienes llevan a cabo la toma de decisiones- a obtener información útil a partir de datos fuente, registros en documentos, conocimiento y experiencia previa. Esto se logra aplicando modelos diseñados de negocio.

Esta información obtenida permite identificar problemas a lo largo de la organización y proporciona una herramienta robusta para poder resolverlos efectuando una correcta toma de decisiones.

El proceso de toma de decisiones dentro de las organizaciones ha tenido significativos cambios en las últimas décadas, desde la aparición del concepto de Sistemas de Soporte a la Decisión (DSS). En un principio, durante la década de los 60´s, de forma exclusivamente académica dentro del ámbito universitario y de investigación.

Este concepto continuó su desarrollo hasta los años 80's donde ya se involucró el diseño de los sistemas de información para ejecutivos (EIS) así como los sistemas de soporte a la decisión organizacionales (ODSS).

En la década de los 90's las tecnologías de procesamiento analítico en línea (OLAP) y Data Warehousing son ya componentes de los DSS. Es en estos años, concretamente hacia finales de los 90's, cuando se incorporan algunas técnicas basadas en aplicaciones web. Podríamos considerar que los DSS aparecieron como una nueva generación dentro del área de sistemas de información, ahora más complejos y sofisticados.

En la actualidad, el uso de los DSS se ha extendido debido a su capacidad de analizar grandes volúmenes de datos y a la forma que tienen de presentar en resumen esta información.

La información que típicamente puede recopilar y mostrar una aplicación de soporte a la decisión incluye todos los datos almacenados en la empresa y que van desde sistemas heredados, hasta bases de datos relacionales o Data Warehouses.

La información que da como resultado el DSS es presentada en esquemas gráficos de tal manera que sean de fácil comprensión aun para los usuarios que no están muy familiarizados con sistemas computacionales. En el DSS de este proyecto, se ha intentado mantener un entorno gráfico que sea capaz de mostrar la información de forma clara y concisa. Por una

parte las soluciones aparecerán dibujadas sobre el mapa, para que cualquier usuario sea capaz de ver rápidamente las soluciones obtenidas.

Una de las funciones más importantes de este tipo de sistemas es la proyección a futuro del comportamiento de algunos factores de negocio, esta proyección está basada en un modelo de sistema que ha sido diseñado de acuerdo con los expertos del área en que se está haciendo la aplicación.

2.3.1 Evolución de los DSS

La tecnología de la información está inmersa en cambios cada vez más rápidos y los DSS no son la excepción. A pesar de su excelente funcionalidad, obviamente como en todos los sistemas siempre habrá algo que mejorar, o bien, errores por eliminar o resolver.

Uno de los principales problemas que presentan en la actualidad los DSS está en su diseño, ya que estos requieren de una considerable experiencia sobre cuestiones estadísticas y de un complejo análisis humano para optimizar sus tiempos de operación (Davenport, Tom. 2006).

Ahora una nueva etapa de sistemas de información que soportan la toma de decisiones está siendo ya desarrollada y aplicada en algunas áreas de negocios y es conocida como Toma de Decisiones Automatizada en una traducción literaria de "Automated Decision Making" (Turban et Al. 2005), considerada como la evolución de los Sistemas de Soporte a la Decisión y de la Inteligencia Artificial (AI), toma las mejores características de ambos para crear el nuevo concepto. Los sistemas de Decisión Automatizada están basados en reglas clásicas de la AI y además involucran conceptos estadísticos o algoritmos de análisis de datos tal como lo hacen DSS tradicionales.

Aunque está todavía en desarrollo, esta tecnología está siendo ya aplicada en áreas como la prescripción médica, servicios de viajes, control del transporte y, también en especial, en el área financiera dentro de empresas bancarias y de seguros.

Algunas de las razones para llevar a cabo este proceso evolutivo es que ahora es mucho más fácil que antes crear y administrar las aplicaciones. Los sistemas de decisión automatizada tienen la capacidad de detectar datos en línea, aplicando lógica o conocimiento codificado, además de tener un fuerte componente de aprendizaje, y tomar decisiones. Todo esto con una mínima participación del elemento humano, que centra su esfuerzo en el diseño y desarrollo de la aplicación.

De este modo muchas decisiones serán tomadas de forma automática por los sistemas y aunque pareciera que solo son buenas noticias para las empresas, ahora el personal gerencial tendrá que afrontar este nuevo reto de la tecnología adaptándose a ella y compartiendo la responsabilidad de ciertos tipos de decisiones.

Igualmente las diferentes economías, sobre todo en países desarrollados, tendrán que resolver algunos aspectos derivados de la pérdida de empleos como producto de la entrada de tecnologías como la de decisiones automatizadas.

2.3.2. Características principales de un DSS

En principio, puede parecer que el análisis de datos es un proceso sencillo, y fácil de conseguir mediante una aplicación hecha a medida o un ERP sofisticado. Sin embargo, no es así: estas aplicaciones suelen disponer de una serie de informes predefinidos en los que presentan la información de manera estática, pero no permiten profundizar en los datos, navegar entre ellos, manejarlos desde distintas perspectivas... etc.

El DSS es una de las herramientas más emblemáticas del Business Intelligence ya que, entre otras propiedades, permiten resolver gran parte de las limitaciones de los programas de gestión. Estas son algunas de sus características principales:

- **Informes dinámicos, flexibles e interactivos**, de manera que el usuario no tenga que ceñirse a los listados predefinidos que se configuraron en el momento de la implantación, y que no siempre responden a sus dudas reales.

- **No requiere conocimientos técnicos**. Un usuario no técnico puede crear nuevos gráficos e informes y navegar entre ellos, haciendo *drag&drop* o *drill through*. Por tanto, para examinar la información disponible o crear nuevas métricas no es imprescindible buscar auxilio en el departamento de informática.

- **Rapidez en el tiempo de respuesta**, ya que la base de datos subyacente suele ser un Data Warehouse corporativo o un Datamart, con modelos de datos en estrella o copo de nieve. Este tipo de bases de datos están optimizadas para el análisis de grandes volúmenes de información.

- **Integración entre todos los sistemas/departamentos de la compañía**. El proceso de ETL previo a la implantación de un Sistema de Soporte a la Decisión garantiza la calidad y la integración de los datos entre las diferentes unidades de la empresa. Existe lo que se llama: *integridad referencial absoluta*.

- **Cada usuario dispone de información adecuada a su perfil**. No se trata de que todo el mundo tenga acceso a toda la información, sino de que tenga acceso a la información que necesita para que su trabajo sea lo más eficiente posible.

- **Disponibilidad de información histórica**. En estos sistemas está a la orden del día comparar los datos actuales con información de otros períodos históricos de la compañía, con el fin de analizar tendencias, fijar la evolución de parámetros de negocio... etc.

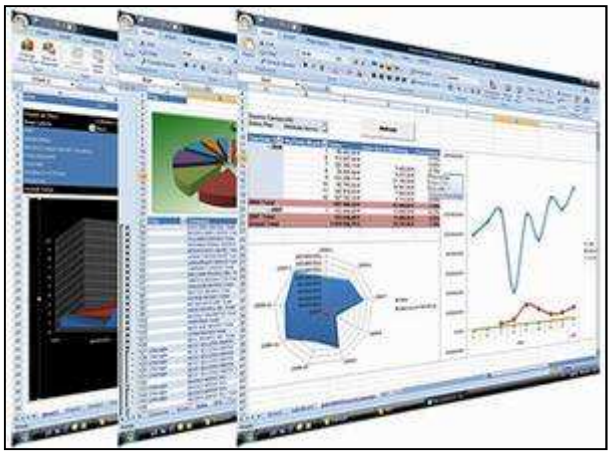


Ilustración 2.6. Ventanas de un DSS comercial.

Diferencia con otras herramientas de Business Intelligence

El principal objetivo de los Sistemas de Soporte a Decisiones es, a diferencia de otras herramientas como los Cuadros de Mando (CMI) o los Sistemas de Información Ejecutiva (EIS), explotar al máximo la información residente en una base de datos corporativa (datawarehouse o datamart), mostrando informes muy dinámicos y con gran potencial de navegación, pero siempre con una interfaz gráfica amigable, vistosa y sencilla.

Ilustración 2.7. Ejemplo de salida de datos de un DSS comercial.

Otra diferencia fundamental radica en los usuarios a los que están destinadas las plataformas DSS: cualquier nivel gerencial dentro de una organización, tanto para situaciones estructuradas como no estructuradas. (En este sentido, por ejemplo, los CMI están más orientados a la alta dirección).

Por último, destacar que los DSS suelen requerir (aunque no es imprescindible) un motor OLAP subyacente, que facilite el análisis casi ilimitado de los datos para hallar las causas raices de los problemas/pormenores de la compañía.

2.3.3. Tipos de Sistemas de Soporte a Decisiones

- **Sistemas de información gerencial (MIS).**

  Los sistemas de información gerencial (MIS, *Management Information Systems*), también llamados Sistemas de Información Administrativa (AIS) dan soporte a un espectro más amplio de tareas organizacionales, encontrándose a medio camino entre un DSS tradicional y una aplicación ERP implantada en la misma compañía.

- **Sistemas de información ejecutiva (EIS)**

  Los sistemas de información ejecutiva (EIS, *Executive Information System*) son el tipo de DSS que más se suele emplear en Business Intelligence, ya que proveen a los gerentes de un acceso sencillo a información interna y externa de su compañía, y que es relevante para sus factores clave de éxito.

- **Sistemas expertos basados en inteligencia artificial (SSEE)**

  Los sistemas expertos, también llamados sistemas basados en conocimiento, utilizan redes neuronales para simular el conocimiento de un experto y utilizarlo de forma efectiva para resolver un problema concreto. Este concepto está muy relacionado con la minería de datos.

- **Sistemas de apoyo a decisiones de grupo (GDSS)**

  Un sistema de apoyo a decisiones en grupos (GDSS, *Group Decision Support Systems*) es "un sistema basado en computadoras que apoya a grupos de personas que tienen una tarea (u objetivo) común, y que sirve como interfaz con un entorno compartido". El supuesto en que se basa el GDSS es que si se mejoran las comunicaciones se pueden mejorar las decisiones.

2.3.4. Ventajas del uso de DSS

Algunas de las ventajas que ofrecen los DSS a una organización son:

- Sistemas interactivos con resultados aplicables a los procesos de negocios.
- Eliminación de intermediarios en el proceso de adquisición de información.
- Reportes personalizados, a la medida del ejecutivo.
- Información expuesta en términos de negocios.
- Análisis de alternativas de acción en entornos controlados.
- Visualización gráfica de los indicadores claves para los distintos procesos de la empresa.
- Facilita el proceso de toma de decisiones y por tanto la productividad de la organización.

Dependiendo del sistema específico de soporte que la empresa u organización desee implementar, se podrán obtener beneficios en lo que se refiere a tiempos optimizados, disminución de costos y beneficios obtenidos en aquellos procesos de negocios a los que se aplique.

2.3.5. Conclusión

Después de dejar en claro que para la alta gerencia de toda organización el llevar a cabo una adecuada y acertada tarea de toma de decisiones como actividad fundamental para asegurar el desarrollo y la competitividad de la misma organización, se entiende entonces que en la actualidad todo tipo de organización busque contar con los mejores sistemas computacionales que ayuden en sus procesos de toma de decisiones.

Sin embargo, es importante que las empresas que deseen implementar un DSS tomen en cuenta que para que dicha implementación resulte exitosa, debe existir un gran apoyo e involucramiento por parte de la alta dirección, así como también una adecuada capacitación para los usuarios finales de la aplicación.

Esto se debe a que un DSS no soluciona problemas, solo apoya en el proceso de la toma de decisiones.

Finalmente la responsabilidad de tomar una decisión, de optarla y de realizarla es de los administradores o de las personas, no del DSS.

3. EL TRANSPORTE POR CARRETERA Y SUS PROBLEMAS

3.1. Evolución y situación actual del transporte

En la actualidad, hablamos de transporte como un concepto abstracto, que abarca multitud de ámbitos de la vida, y que asumimos con total naturalidad, al igual que sus consecuencias. Tanto los vehículos como las infraestructuras necesarias para dicha actividad, forman parte del paisaje cotidiano de las ciudades, del mismo modo que el hecho de viajar, sean cuales sean las causas, a cualquier lugar del mundo, ya no sorprende a nadie. Todo este desarrollo en materia de transporte aporta incalculables ventajas a las sociedades modernas, pero también supone un coste extra para la salud humana y el equilibrio natural. El gran problema es debido al masivo incremento de esta actividad, siendo éste más rápido que las medidas correctoras que pudieran implantarse, por lo que nos encontramos en una situación de desequilibrio que de alargarse en el tiempo, tiene visos de convertirse en insostenible.

Si hablamos en los términos de los objetivos de este proyecto, trataremos principalmente del transporte por carretera y de sus consecuencias. Históricamente, las carreteras han resultado imprescindibles para la creación del espacio económico de los diferentes países. Es el modo de transporte de personas y productos más utilizado, ya que el transporte por carretera supone cerca del 90% del total, a nivel global. Su existencia supone un agente de transformación social, económica y paisajística nada despreciable (*Foro medioambiental del transporte por carretera*, 2003).

A mediados del siglo pasado, las necesidades de transporte no eran tan grandes como en la actualidad. El diseño y trazado de las infraestructuras dependía en gran medida de la morfología del terreno y su orografía. Con unos recursos de maquinaria de movimiento de tierras algo limitada, unida a un gran número de mano de obra disponible, la elección de alternativas estaba enormemente condicionada. Como resultado de todo esto surgen unas infraestructuras estrechas, adaptadas a la geomorfología del terreno y con una alta siniestralidad. Estas carreteras suponían unas vías de comunicación interurbanas de importante transcendencia para aquel momento. En muchos casos, el trazado de las mismas dividía los núcleos urbanos en dos, formando las travesías urbanas, tan criticadas y problemáticas en la actualidad. Estas carreteras, y en particular los tramos de travesía suponían un auge en la economía de la zona por la que discurría la carretera, sobre todo en los sectores secundario y terciario, debido al transcurso de un gran número de vehículos por el núcleo urbano (*Seguí y Petrus*, 1991).

En los últimos años, el modelo productivo y de transformación ha supuesto un cambio muy grande en un corto espacio de tiempo. La globalización actual, ha traído consigo procesos de localización y deslocalización de actividades productivas a nivel mundial, que unida al aumento exponencial de la población, ha ocasionando un auge considerable de todos los sectores productivos y especialmente del transporte. Las sociedades de consumo actuales cada vez

demandan más productos y se da este distanciamiento geográfico entre el producto recién fabricado y su consumidor final. Es aquí donde entra en juego el sector logístico, cuyo objetivo es conseguir transportar el producto desde su lugar de origen hasta las manos del consumidor, tratando siempre de minimizar los costes.

Según la AEMA1 (*Agencia Europea de Medio Ambiente*), el transporte representa cerca de la tercera parte de todo el consumo final de energía en sus países miembros, lo que supone más de una quinta parte de las emisiones de gases de efecto invernadero. El transporte es responsable también de un elevado porcentaje de la contaminación del aire en las zonas urbanas, así como de las molestias causadas por el ruido. Además, el efecto barrera provocado por las infraestructuras lineales divide las zonas de interés ecológico, dando lugar a problemas de incomunicación entre áreas naturales, con graves consecuencias para la fauna e incluso la flora. (*www.eea.europa.eu/es/themes/transport*)

Siguiendo en el ámbito de la Unión Europea, el volumen de tráfico de mercancías está creciendo a una tasa incluso superior a la del crecimiento económico por los cambios en los modelos de producción y distribución. Y el informe de la Comisión Europea (EU15 Energy and Transport Outlook to 2030) prevé que la demanda de transporte de mercancías continúe creciendo a una tasa del 2,1% anual en los próximos veinte años.

Según este informe, la carretera es el modo que experimenta un mayor crecimiento, pues se han pasado de transportar así el 52,1% del total de mercancías en 1970 (488 billones de toneladas-km) a transportar el 75,5% en 2001 (1.396 billones de tn-km).

En contrapartida, ha disminuido la importancia del ferrocarril, que apenas mueve el 8% de la mercancía transportada. No obstante, entre los estados miembros se observan algunas divergencias en los tipos de transporte utilizados, que se explican por la diversidad geográfica, por diferencias en las mercancías transportadas y por las distintas políticas económicas puestas en práctica por cada uno de los gobiernos.

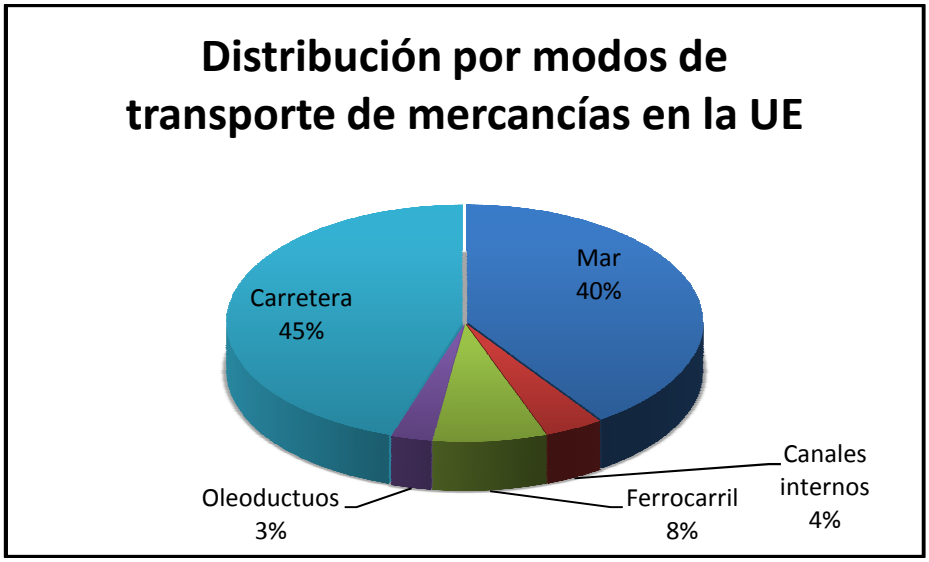


Gráfico 3.1 European Foundation for the Improvement of Living and Working Conditions (2004).

Además de los citados con respecto al crecimiento general del sector, otros factores que explican el incremento del peso relativo de la carretera son:

- El incremento de los ingresos, que se traduce en un aumento de la capacidad adquisitiva y de la propiedad de vehículos.
- Las inversiones en infraestructuras, donde se han priorizado las carreteras por parte de los gobiernos.
- Las mejoras tecnológicas, que generan menores precios y mayores velocidades de los vehículos.
- El empeoramiento de la calidad del transporte público por ferrocarril.

En España, en 2005, el transporte ocupó a 938.526 personas, de las que 730.521 (77,8%) eran asalariados y generó un valor añadido de 34.178 millones de euros, lo que supone el 6,25% del VAB del sector servicios y el 3,8% del PIB nacional a precios de mercado (INE: Encuesta Anual de Servicios 2005; INE: Contabilidad Nacional de España, 2005). Según los últimos datos disponibles (Roza Braga 2007), la carretera supone el 80,3% del consumo final del sector seguida del transporte aéreo (12,5%), marítimo (4,3%) y ferroviario (2,9%).

Centrándonos en la situación de los Pirineos, en 2008, el tráfico de mercancías entre la Península Ibérica y los demás países comunitarios representaba ya 144 millones de toneladas, 63 de las cuales (el 44%) circulaban por vía marítima, y 81 (es decir, el 56%), por vía terrestre. De estas últimas, no llegan a 4 millones de toneladas (lo que representa poco más del 4% del total), las correspondientes al transporte ferroviario, lo quiere decir que el porcentaje de transporte por carretera en los intercambios terrestres a través de los Pirineos es superior al 95%. Esto se traduce en más de 15.000 camiones al día por los pasos pirenaicos (frente a los 8.300 que soportan los Alpes), dando una idea del grado de asfixia en el que se encuentran los Pirineos. (*Fundación Transpirenaica* 2003-2010)

Si tenemos en cuenta de las perspectivas del Observatorio de tráfico del Pirineo, lo peor está aún por llegar. Para el año 2020, aplicando una tasa de crecimiento similar a la experimentada estos últimos años, el tráfico de mercancías podría verse incrementado en 100 millones de Toneladas adicionales, lo que equivaldría a unos 250 millones de Toneladas en total, y más de 30.000 camiones atravesando los Pirineos a diario, si no encuentran alternativas mejores.

Esta saturación, se localiza prácticamente en dos principales pasos por carretera de la frontera pirenaica, puesto que entre ambos abarcan el 94% del tráfico por carretera. Existen otros tres pasos más, pero mucho más modestos. Los dos principales pasos de los Pirineos son el de La Junquera (Girona) y el de Irún (Guipúzcoa).

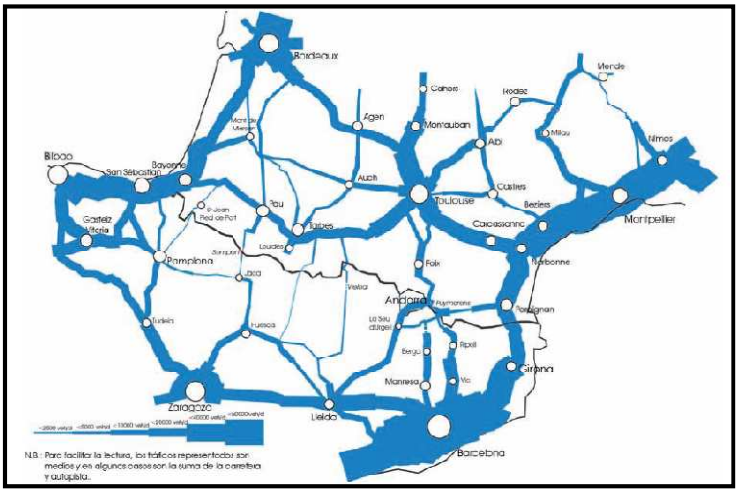


Ilustración 3.1. Intensidad Media Diaria del tráfico rodado a través de los pirineos. (Ministerio del medio ambiente, energía y desarrollo de Francia.

En Navarra existen pequeños pasos para cruzar la frontera, pero principalmente es la cercanía con Irún la que hace que gran parte del tráfico proveniente de la Península Ibérica atraviese el territorio navarro para seguir sus rutas hacia el resto de Europa.

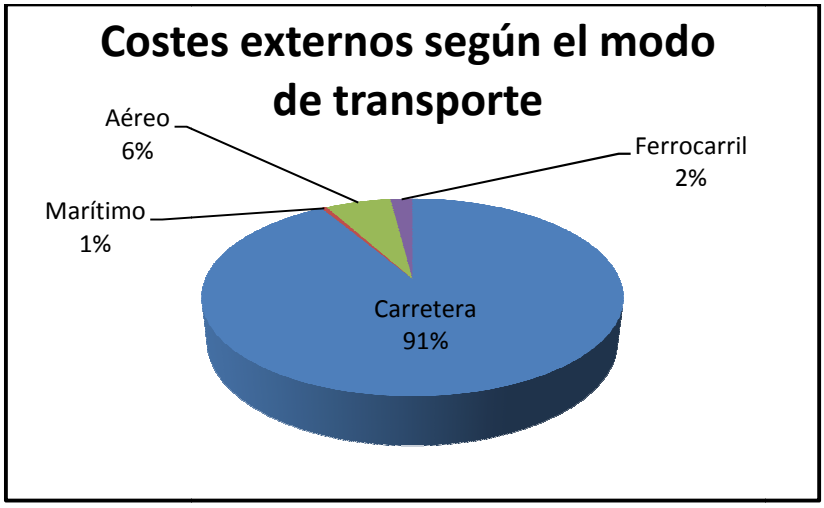
3.2. El transporte desde el punto de vista del proyecto

Hasta aquí, hemos hablado del transporte en general, así como de los efectos negativos que tiene en la población y en el medio ambiente. A partir de ahora, vamos a hablar del transporte, pero centrándonos en aquellas facetas del mismo que más importancia tienen para este proyecto.

3.2.1. Transporte de mercancías por carretera

Anteriormente se ha comentado de forma general la situación actual del transporte en Europa, en España y concretamente en el Pirineo. Debido al creciente peso del transporte y a su proximidad a la Comunidad Foral de Navarra, en este proyecto nos hemos centrado en el estudio del transporte por carretera y concretamente, en el transporte de mercancías por carretera. A diferencia del transporte de personas, el transporte de mercancías se trata de una actividad que requiere de la logística, por lo que permite realizar tareas de planificación mediante las cuales alcanzar los objetivos de reducción de emisiones que se persiguen. Otra razón para fijarnos en el transporte de mercancías y no en el de personas, es la gran dificultad que supondría el tratar de organizar y planificar las rutas de los transportes de viajeros, principalmente los particulares. Esto sería perseguir un objetivo poco realista e imposible de poner en práctica con los medios actuales, además de interferir en el derecho de libre movimiento.

Además, si atendemos a los costes externos provocados por los distintos modos de transporte, podemos apreciar que el porcentaje correspondiente al transporte por carretera es superior al 90 % del total. Este hecho constituye otra razón más para centrarnos en esta modalidad de transporte y da una visión de la necesidad existente de reducir dichos costes externos.



S/IWW (2000)

Como ya se ha explicado con anterioridad, en el presente proyecto se crea una herramienta informática de apoyo a las decisiones, que ha de ser capaz de mitigar alguna de las

consecuencias negativas derivadas del transporte de mercancías por carretera, mediante el diseño de rutas sostenibles.

Se pretende que esta herramienta resulte de utilidad en las primeras fases del proceso de transporte, concretamente, una vez se conocen todos los clientes y sus demandas y llega el momento de planificar una ruta de reparto.

Por tanto, la idea central es poder diseñar unas buenas rutas, que sean lógicas desde el punto de vista de los costes y que además sean respetuosas con el medio y la población. Hay que recordar que siempre será un paso previo al propio acto del transporte, y que una vez éste se inicia, el DSS ya no tendría utilidad práctica, aunque si puede ser utilizado como medio de simulación y de cálculo de costes.

3.3.2. Fases del transporte de mercancías

Una vez elegido la modalidad de transporte con la que se va a trabajar, el transporte de mercancías por carretera, he creído conveniente el desarrollo de un modelo simplificado del mismo. El transporte de mercancías se va a tratar como un proceso sencillo, que se inicia con la demanda por parte del destinatario o cliente de unos bienes y concluye cuando los vehículos de entrega regresan al punto de partida una vez satisfechas todas las demandas. Esto no es más que una simplificación de una tarea tan compleja como la del transporte y la distribución, pero va a permitir establecer de forma clara los objetivos que se pretenden conseguir con el desarrollo del DSS.

A partir de ahora, se seguirá el siguiente modelo simplificado del proceso del transporte de mercancías:

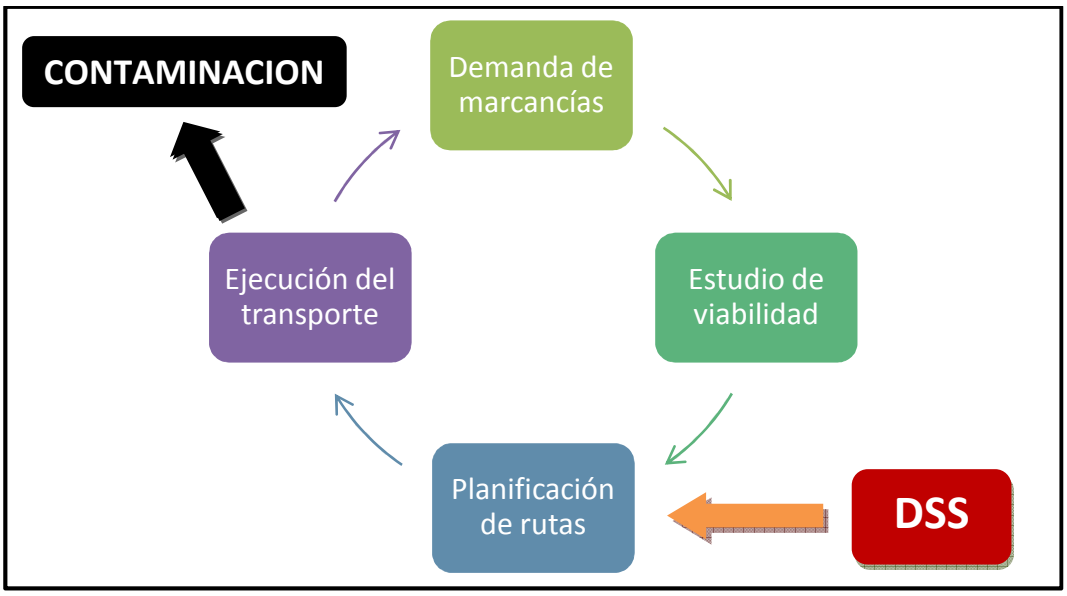


Ilustración 3.2 Esquema de las fases del transporte. (Elaboración propia)

Cuando es recibida una petición de mercancía por parte de un cliente, el suministrador decide si puede atenderla o no. Para la toma de esta decisión, el suministrador debe tener en cuenta factores como la disponibilidad de la mercancía en el almacén o la situación de los vehículos en ese momento. Además tiene que intentar predecir cómo afectará la inclusión del nuevo cliente en las rutas ya establecidas. Esta fase del proceso de transporte también puede ser tratada con un DSS, ya que podría ser de gran utilidad para mantener y actualizar la información referente a la gestión de almacenes y de stocks, y recomendar cuando es más conveniente la atención de una petición de un cliente. Sin embargo, este no es el objetivo del presente proyecto, que se centra en la siguiente fase del proceso, la planificación de rutas.

Si finalmente se decide atender la nueva petición, se inicia entonces una serie de tareas cuyo fin es la entrega del producto al nuevo cliente, en forma y plazo correctos y al menor coste posible. Entre estas tareas está la de planificar una ruta de reparto que permita atender a todos los clientes con el menor número de recursos posible.

Es aquí donde el DSS de este proyecto entra en juego. Como se explicará en los próximos capítulos, el DSS es capaz de, conocidos unos clientes y sus demandas, encontrar rutas eficientes de reparto, minimizando los costes. Si bien hasta ahora minimizar costes significaba reducir los kilómetros recorridos y las horas empleadas en ello, el DSS tiene en cuenta los otros costes derivados del transporte. Concretamente este DSS trata de minimizar el impacto de las rutas elegidas en la población y en el medio ambiente.

Una vez calculada la ruta de reparto, se inicia la ejecución del transporte. Esta parte engloba las tareas logísticas correspondientes para poner a punto todos los vehículos necesarios, cargarlos y enviarlos a seguir la parte de la ruta correspondiente a cada uno. Una vez vacíos, los vehículos deberán regresar al almacén, bien para ser cargados de nuevo y continuar el reparto, o bien para esperar hasta próximas peticiones y por tanto, nuevas rutas que seguir.

Durante la fase de ejecución del transporte es donde se produce la mayor parte del impacto negativo del mismo, concretamente se producen las emisiones de partículas contaminantes a la atmósfera, así como una incidencia importante en los niveles acústicos, perceptible por las personas que viven o transitan cerca de las vías y carreteras. Todas estas emisiones nocivas son principalmente la consecuencia residual del funcionamiento de los motores de explosión de los vehículos terrestres, así como de su propia rodadura sobre el asfalto.

Hoy en día, con la tecnología disponible y a pesar de los últimos avances, aun no es posible eliminar estas emisiones ni sustituir los motores de explosión por otros más respetuosos, por lo que el esfuerzo realizado en este proyecto se centra en reducir los efectos nocivos de la actividad del transporte seleccionando rutas que eviten, siempre que sea posible, las zonas de especial interés ecológico o los núcleos de población.

Como consecuencia del continuo aumento de las actividades del transporte, si bien se ha conseguido un nivel de desarrollo económico sin precedentes, se producen una serie de consecuencias en el medio ambiente y en la calidad de vida de las personas, ya que modifica la morfología del terreno y vierte a la atmósfera grandes cantidades de gases peligrosos por su toxicidad. Como veremos más adelante, los esfuerzos de investigación en este campo se enfocan en limitar el alcance o compensar los efectos negativos del transporte, con el fin de mantener el ritmo de crecimiento y desarrollo actual de un modo sostenible.

3.3. Problemas derivados del transporte

Las directivas europeas publicadas por el Parlamento Europeo tienen cada vez una mayor consideración por la población que sufre altos niveles de contaminación acústica y atmosférica, entre otras. Cada uno de los estados miembros y sus comunidades, publican su propia normativa que intenta, como en caso de la contaminación acústica, mantener a la población dentro de un rango de confort sonoro aceptable, definiendo unos umbrales que son perjudiciales para la salud y el confort de las personas.

En el caso de España, el Real Decreto 1367/2007 por el que se desarrolla la Ley 37/2003 del ruido, en lo referente a zonificación acústica, objetivos de calidad y emisiones acústicas. Para corregir ese déficit de calidad ambiental, las administraciones editan sus propios mapas estratégicos de contaminación acústica, para realizar pequeños proyectos locales, como la colocación de barreras acústicas, que palien ese efecto y reducir así el nivel sonoro al que están expuestas las personas que viven en las proximidades de las vías de transporte. Otro ejemplo claro y reciente, es la Ley 34/2007, de calidad del aire y protección atmosférica, que establece las bases de prevención, vigilancia y reducción de la contaminación atmosférica, con el fin de aminorar los daños que de ésta puedan derivarse para las personas, el medio ambiente y demás bienes de cualquier naturaleza.

Así como ocurre con la contaminación acústica y atmosférica, podríamos hacer un paralelismo con el efecto barrera que ocasionan las infraestructuras lineales de transporte. A pesar que este impacto ambiental no tenga una legislación como tal, muchos son los proyectos que intentan mejorar los accesos a las áreas limítrofes con las carreteras, y los numerosos pasos a nivel que están siendo eliminados por el alto grado de siniestralidad que los caracteriza, construyendo puentes y viaductos que permitan un mejor comunicación y acceso a las zonas colindantes con las infraestructuras. Este tipo de proyectos, por lo general, se realizan en infraestructuras lineales con una cierta antigüedad, y son medidas correctoras que permiten recuperar o paliar parte del impacto que provocaba esa infraestructura primando la seguridad y el confort de las personas.

Todas estas consecuencias negativas del transporte pueden clasificarse como costes derivados del transporte, y de esta forma puede cuantificarse el daño que provocan. Además permite establecer una forma de comparar los costes tradicionalmente asociados al transporte con los costes derivados de la actividad, de manera que se puedan buscar soluciones para compensar a los miembros perjudicados y alcanzar un equilibrio real.

3.3.1. Costes del transporte por carretera. Externalidades.

Tradicionalmente se habla de costes del transporte como aquellos derivados del uso directo del vehículo, como pueden ser el combustible, el mantenimiento, las ruedas… que forman parte de la propia actividad del transporte y por tanto son asumidos por el sujeto que ejecuta dicha actividad, ya sea una empresa o un particular.

Históricamente, los esfuerzos por reducir costes derivados del transporte se han orientado a intentar minimizar estos costes directos, sin tener en cuenta otro tipo de factores. En el momento actual, en muchos países industrializados y concretamente en la Unión Europea, se ha empezado a considerar los efectos negativos que provoca esta actividad tanto en la población como en el medio ambiente, y se están empezando a promover medidas dirigidas a internalizar estos costes. Es por eso que uno de los objetivos de éste proyecto consiste en tener en cuenta los costes soportados por terceros derivados de la actividad del transporte, lo que se conoce como costes externos o externalidades.

Se entiende como externalidad o efecto externo los cambios causados en el bienestar de aquellos individuos que no intervienen directamente en esas actividades, que son externos a esa producción o consumo (*Barrios et al.,* 2003). Las externalidades se producen cuando las actividades sociales o económicas de un grupo de personas tienen un impacto sobre otras personas y dicho impacto no está plenamente tomado en cuenta por el primer grupo.

El impacto debido a las externalidades del transporte se fundamenta en los siguientes cuatro elementos:

1. Impacto sobre la salud humana.
2. Daños causados a los ecosistemas locales:
    a. Impacto en las cosechas y producción agrícola
    b. Daño a los bosques
3. Daños materiales causados a edificios y bienes.
4. Calentamiento global y cambio climático.

Otra forma de entender las externalidades es como eventos que confieren beneficios o costes considerables a una persona o grupo de personas sin que éstas hayan dado su consentimiento en el momento de tomar las decisiones que llevaron directa o indirectamente a la concurrencia de dichos eventos. Es decir, una externalidad se presenta cuando la utilidad del individuo A se ve afectada por ciertas variables cuyos valores son decididos por otros (personas, empresas o gobiernos), sin interesarse en los efectos y el bienestar del individuo A.

Concretamente, en el caso que nos ocupa, la actividad económica es el transporte de mercancías por carretera, que constituye en sí mismo un medio de negocio, y como terceros

se incluye el medio ambiente (flora y fauna) además de la población humana. Las externalidades se dan con frecuencia en actividades relacionadas con el medioambiente, en casos en los que los derechos de propiedad no están bien definidos. Las soluciones que se aplican en la realidad suelen comprender tanto los impuestos y las subvenciones como la regulación.

La existencia de estas externalidades implica un desequilibrio en el mercado. Es aquí donde los gobiernos han de intervenir para aumentar la eficiencia del mercado y recuperar el equilibrio perdido. De este modo, la Unión Europea está estudiando la opción de introducir medidas correctoras como forma de compensación de las externalidades producidas por el sector, consiguiendo así la internalización de los costes externos y alcanzando de nuevo el equilibrio del mercado.

En general, las externalidades medioambientales asociadas al transporte en el ámbito europeo constituyen una variable significativa en términos de PIB. Diversos estudios han elevado la estimación correspondiente de las externalidades en el PIB en la Unión Europea más Suiza y Noruega, desde un 4,6% en 1995 hasta el 8,4% en el año 2000, teniendo en cuenta los costes de congestión (*INFRAS/IWW*, 2000).

Para el caso de España, se estima que el medio más impactante sobre el medioambiente es el transporte por carretera, que origina el 83,7% del coste total. En el período 1995-2000 los costes totales aumentaron un 12,1% (valores de 1995 ajustados a precios de 2000). La principal razón de esta evolución radica en el crecimiento de los volúmenes de tráfico que han dado lugar a un mayor volumen de emisión de gases de efecto invernadero y, por consiguiente, a un incremento de los riesgos de cambio climático. Aunque las emisiones de partículas procedentes de gases de combustión han descendido de forma importante gracias a la mejora de la tecnología de los motores y a los filtros de partículas, las emisiones de partículas no relacionadas con la combustión han aumentado, aproximadamente, de acuerdo con los volúmenes de tráfico.

Por ello, si bien el transporte contribuye de manera muy importante al desarrollo económico y social, su enorme crecimiento en los últimos años hace que su impacto en la salud de las personas y el medioambiente también haya crecido a la par. En presencia de costes externos, el mercado presenta precios erróneos lo que conlleva a un derroche de recursos escasos. Para poder aplicar las políticas correctoras a esta situación de ineficiencia del mercado, la estimación de esos mismos costes externos se muestra fundamental.

En el ámbito del transporte y la logística, la consideración de las externalidades ha modificado la política de transporte, especialmente en la Unión Europea. De este modo, la valoración económica de las externalidades negativas es considerada una herramienta esencial para incluir los costes ambientales y sociales en la toma de decisiones políticas y empresariales, que eviten un despilfarro de recursos escasos y vitales. Se plantea la posibilidad de internalizar los

impactos externos mediante algunas herramientas como son la asignación de los derechos de propiedad, impuestos y subvenciones, mecanismos de compensación y la regulación en el uso de las infraestructuras.

En consecuencia, la necesidad de un conocimiento acerca de cómo hay que realizar un transporte y unas comunicaciones que sean compatibles con un impacto reducido sobre el medioambiente, se revelan de capital importancia. Por otra parte, el impacto medioambiental del flujo de transporte en las zonas transfronterizas dentro de la Unión Europea se presenta como un aspecto muy importante a conocer dentro de los estudios de carácter europeo.

3.3.2. Compensación de las externalidades

La situación de fallo de mercado que se produce con los impactos externos, abre las puertas a la intervención del sector público. La teoría económica sugiere hacerlo mediante los siguientes instrumentos correctores:

- **Asignación de los derechos de propiedad**: Las externalidades se caracterizan por una situación en la que no están bien definidos los derechos de propiedad.
- **Impuestos y subvenciones**: Fijación, por parte del sector público, de impuestos (caso de externalidad negativa) o subvenciones (caso de externalidad positiva) que reflejen la valoración marginal de los efectos externos y permitan su internalización.
- **Regulación**: El sector público establece normas legales que fijen el nivel óptimo de producción o consumo en presencia de externalidades.
- **Mecanismos de compensación**: La aplicación de impuestos y subvenciones, así como de la regulación, plantea problemas ya que exige al sector público conocer la valoración marginal de los efectos externos por parte de los agentes.

La OCDE (2002), en sus directrices para un transporte ambientalmente sostenible, establece la necesidad de incorporar los costes externos en el análisis de viabilidad económica e implicaciones de la política de transporte. De forma similar, la Conferencia Europea de Ministros de Transporte instó en el año 1998 (*CEMT/OM (98)5/Final*) a los gobiernos a desarrollar instrumentos para la internalización de las externalidades negativas en el transporte.

Más recientemente, la Comisión Europea ha incorporado este enfoque en la política comunitaria de transportes mediante el Libro Blanco sobre Tarifas Justas por el Uso de las Infraestructuras y el Libro Blanco del Transporte (*Comisión Europea*, 1998b y 2001). Más recientemente, en el año 2004, el Parlamento Europeo solicitaba a la Comisión la creación de un modelo de aplicación general, transparente y comprensible para la valoración de todos los

costes externos ambientales, de congestión y relacionados con la salud, que sirva de base para cálculos futuros de tarificación de infraestructuras (*Parlamento Europeo*, 2004).

Fue a partir de septiembre de 2001, cuando con el Libro Blanco del Transporte (*Comisión Europea*, 2001) la Comisión Europea planteó como principales problemas del sector de transporte en Europa las limitaciones de las infraestructuras, la congestión y los accidentes, y como principal instrumento para afrontarlos, la tarificación del uso de infraestructuras, internalizando los costes externos no soportados por los usuarios del sistema. Esto es lo que se conoce como la directiva Euroviñeta (*Comisión de las Comunidades Europeas*, 2003).

Asimismo, la Comisión Europea insta a los países miembros a incluir los costes medioambientales en el cálculo de dichas tasas de peaje, variando el valor del peaje en función del nivel de emisiones del vehículo.

En cualquier caso, el proceso de cálculo económico de los costes externos, y de establecimiento de procedimientos de recaudación, tiene un gran número de incertidumbres, aunque se hace necesario mejorar la gestión de la oferta y la demanda de transporte, de manera que permita recuperar el equilibrio en el sector.

A modo de resumen, entre los costes externos estudiados en el transporte, cabe destacar los costes por cambio climático, coste de accidentes, de ruido, de la contaminación del aire, de congestión, costes para la naturaleza y el paisaje y costes adicionales en áreas urbanas, además de otros efectos indirectos (*INFRAS*, 2006).

Para la realización del DSS tendremos en cuenta sólo dos de estos costes externos, la emisión de partículas a la atmósfera y la contaminación acústica.

La contaminación debido a las emisiones de partículas y de CO_2 es de una gran importancia, ya sea como efecto dañino a la salud de las personas o como impacto en medios naturales y artificiales. Entendiendo como medios naturales los bosques y cultivos, mientras que los artificiales se refieren a los edificios y otras construcciones realizadas por las personas.

En comparación, la contaminación acústica puede resultar de menor incidencia, si bien no hay que olvidar que la mayoría de los estudios se realizan en entornos urbanos, y el ruido existente en las ciudades debido a múltiples factores puede encubrir el coste debido al ruido.

En los siguientes apartados se detalla la contaminación acústica y la atmosférica, como preludio de la estimación del impacto generado por las mismas y su posterior estudio en el DSS. Estos dos costes externos serán los criterios medioambientales del DSS.

3.4. Criterios medioambientales del DSS

Si bien se ha explicado que existen multitud de elementos afectados por las emisiones del transporte, he tenido que escoger cuáles de ellos son susceptibles de mejorar su situación a través de modificaciones en las rutas de circulación. No podemos evitar la emisión de gases a la atmósfera, pero si podemos intentar que sean emitidos lo más lejos posible de zonas críticas por su sensibilidad. En este sentido, tal y cómo se hizo en el proyecto TRANSPIR, se han escogido las emisiones de partículas a la atmósfera y la contaminación acústica como los dos agentes contaminantes más directos y más fácilmente reconocibles por las personas.

Una vez escogidos estos dos factores contaminantes (emisiones de partículas y ruido) como los objetivos de reducción principales, se estudió cómo estos afectaban tanto a las personas como al medio ambiente. De estos estudios y de la valoración subjetiva de los individuos se llega a la conclusión de que cuanto más cerca se encuentra el individuo de la fuente emisora, mayor es la incidencia negativa sobre él. Esta idea es, en principio, igual de válida para personas y para zonas naturales.

De esta idea parte la hipótesis en la que se basa el DSS: Si conseguimos alejar las rutas de reparto de las zonas naturales de especial interés y logramos reducir el número de personas afectado por dichas rutas, lograremos, en general, que el daño generado por la ruta sea menor en términos globales.

3.4.1. Contaminación acústica

Se llama contaminación acústica (o contaminación auditiva) al exceso de sonido que altera las condiciones normales del ambiente en una determinada zona. Si bien el ruido no se acumula, traslada o mantiene en el tiempo como las otras contaminaciones, también puede causar grandes daños en la calidad de vida de las personas si no se controla adecuadamente.

De forma más general, el término contaminación acústica hace referencia al ruido (entendido como sonido excesivo y molesto), que normalmente es provocado por las actividades humanas (tráfico, industrias, locales de ocio, aviones, etc.), que produce efectos negativos sobre la salud auditiva, física y mental de las personas.

El término “contaminación acústica” está estrechamente relacionado con el concepto de “ruido”, debido a que ésta se da cuando el ruido es considerado como un contaminante, es decir, un sonido molesto que puede producir efectos nocivos fisiológicos y psicológicos para una persona o grupo de personas.

Desde el punto de vista físico, el sonido, y por lo tanto el ruido, es una vibración del medio, una onda mecánica que se genera y propaga a través de un medio elástico (gases, líquidos o sólidos) con una intensidad y frecuencia determinada, provocando una vibración acústica

capaz de producir una sensación auditiva. La intensidad del sonido corresponde a la amplitud de la vibración acústica, la cual es medida en decibelios (dB). El ruido ha sido definido desde el punto de vista físico como una superposición de sonidos de frecuencias e intensidades diferentes, sin una correlación de base.

Se ha advertido por parte de internacionales, como es el caso de la Organización Internacional de la Salud (OMS), que se corre el riesgo de una disminución importante en la capacidad auditiva, así como la posibilidad de trastornos que van desde lo psicológico (paranoia, perversión) hasta lo fisiológico por la excesiva exposición a la contaminación acústica.

En el caso del ruido, se conoce que su difusión y alcance están relacionados con la distancia desde la fuente emisora hasta el receptor. Este hecho, asimilado por todos como algo natural, hace que sea interesante el buscar rutas que no atraviesen ni pasen cerca de los núcleos de población, siempre y cuando esto sea posible.

Hoy en día, los órganos de gobierno, ya sean a nivel municipal o autonómico, están tomando medidas a este respecto, mediante la modificación de las infraestructuras y la creación de otras nuevas, como pueden ser las variantes y circunvalaciones que rodean las poblaciones de un cierto tamaño. El problema reside en que aún quedan muchos núcleos de menor tamaño que son atravesados directamente por las carreteras, y en algún caso con grandes volúmenes de tráfico.

Otras medidas que se están tomando en casos más puntuales, es la colocación de barreras acústicas entre las vías y las zonas residenciales. De este modo se logran suavizar las ondas sonoras que llegan hasta las viviendas, aunque no se reducen por completo.

Algo tan subjetivo como el ruido es difícil de medir si no se cuenta con los instrumentos necesarios. En la siguiente tabla se muestran diferentes niveles sonoros junto con una actividad cotidiana que los representa. De este modo es más fácil comprender de qué valores límite se habla en las diferentes legislaciones existentes a este respecto.

Nivel de Presión Sonora dB(A)	Sensación Acústica	Ejemplos
< 0	No audible	Cámara anecoica
0	Umbral de audibilidad	Teste de audiometría
10	Muy silenciosa	Estudio de grabación
20		Grutas
30	Silenciosa	Dormitorio
40		Oficina tranquila
50	Moderada	Oficina
60	Modesta (para un trabajo intelectual)	Conversación a 1 metro
70	Moderadamente desagradable	Calle peatonal – taller
80	Desagradable	Estación de tren
90	Umbral de peligro si > 8 horas diarias	Taller con maquinaria
100	Muy fuerte	Maquinaria de laminado
110	Los gritos no son audibles	
120	"Sordera"	
130	Umbral de dolor	Avión despegando

Tabla 3.1. Niveles de ruido y sensación acústica subjetiva.

Los primeros requisitos armonizados europeos sobre el ruido de los vehículos de carretera se introdujeron en 1970 con la Directiva 70/157/CEE relativa a la aproximación de las legislaciones de los Estados Miembros sobre el nivel sonoro admisible y el dispositivo de escape de los vehículos a motor. La misma se ha modificado varias veces posteriormente para revisar y hacer más estrictos los límites de ruido homologados como parte del marco europeo de homologación de los vehículos de motor. Los límites vigentes ahora figuran en la Tabla 2.6. A escala internacional, el Foro Mundial para la Armonización de las Reglamentaciones aplicables a Vehículos en la Comisión Económica de las Naciones Unidas para Europa (CEPE/ONU) ha elaborado el Reglamento nº 51 sobre las emisiones sonoras de los vehículos de carretera, que se considera equivalente a la Directiva 70/157/CEE.

No obstante, pese a que los límites de ruido homologados se han vuelto más estrictos año tras año, no se ha introducido ninguna mejora en lo relativo a la exposición global al ruido generado por los vehículos de carretera. El crecimiento y la expansión del tráfico en el espacio y en el tiempo, y el desarrollo de las actividades de ocio y turismo han anulado parcialmente los efectos de los avances tecnológicos.

Tipo de vehículo de motor	Límite (dB(A))
Vehículos destinados al transporte de personas y con nueve asientos como máximo incluido el asiento del conductor.	**74**
Vehículos destinados al transporte de personas, con más de nueve asientos incluido el asiento del conductor, una masa máxima autorizada de más de 3,5 toneladas y:	
- una potencia del motor inferior a 150 kW.	**78**
- una potencia del motor de 150 Kw. o más.	**80**
Vehículos destinados al transporte de personas con más de nueve asientos incluido el asiento del conductor, y vehículos destinados al transporte de mercancías:	
- con una masa máxima autorizada que no supere las dos toneladas.	**76**
- con una masa máxima autorizada de entre 2 y 3,5 toneladas.	**77**
Vehículos destinados al transporte de mercancías, con una masa máxima autorizada superior a 3,5 toneladas y:	
- potencia del motor inferior a 75 kW.	**77**
- potencia del motor de 75 Kw. o más, hasta 150 kW.	**78**
- potencia del motor de 150 Kw. o más.	**80**

Tabla 3.2: Límites vigentes en la Unión Europea

3.4.1.1. Dispersión del ruido

La dispersión geométrica de la energía sonora en el espacio es la resultante de la propagación de la onda acústica, que siempre sufre una atenuación en el nivel sonoro cuando se aumenta la distancia desde el receptor hasta el observador.

Cuando se considera que la dispersión es esférica (todas direcciones) la disminución del ruido al hacer que la distancia sea el doble la superficie del frente de onda se hace cuatro veces mayor, y la presión sonora disminuye 6dB (*Embleton*, 1996). En cambio si se considera la propagación del ruido en forma cilíndrica, la emisión del ruido es a lo largo de un eje, y en este caso el área del cilindro es proporcional con la distancia, y por eso la disminución del nivel de ruido es de 3 dB doblando la distancia.

En el segundo caso, si consideramos un receptor cercano de la carretera, el nivel de ruido que recibe dependerá casi exclusivamente del vehículo que circula en frente suya, dado que el resto están relativamente lejanos. Este sería el caso de un vehículo pasando solo por la carretera. Por tanto, un receptor cercano a la carretera se verá afectado por un nivel de ruido fluctuante entre el máximo y el mínimo. Si el receptor se aleja de la carretera, el resto de vehículos empieza a jugar un papel importante, ya que las distancias entre el receptor y las diferentes fuentes emisoras son comparables.

Es por eso que se realizan las medidas de lo que se conoce como nivel sonoro equivalente que se define de la siguiente manera:

$$L_{eq} = 10 \log\left[\left(\frac{1}{T}\sqrt{\int_0^T p_A{}^2(t)dt}\right) / p_{ref}^2\right]$$

Nivel sonoro continuo equivalente

T tiempo promedio

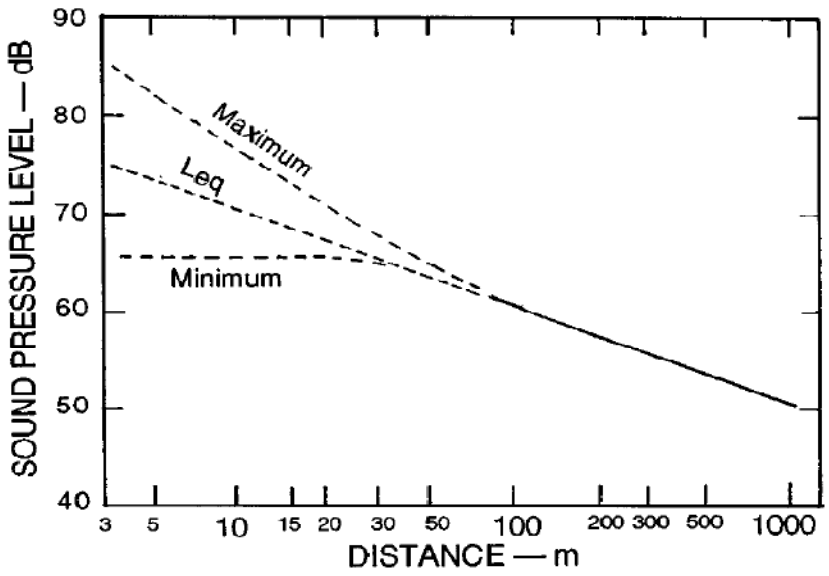


Gráfico 3.3. Atenuación del nivel de ruido con la distancia *(Embleton, 1996)*

Como se explica en la introducción, en este proyecto se aprovecha la experiencia y la información obtenidas durante la colaboración en el proyecto TRANSPIR. En dicho proyecto se utilizaron varias herramientas software que simulan la dispersión del ruido y de las partículas.

En el caso del ruido, este software fue el Custic 2.0, una herramienta capaz de predecir cómo se ha de comportar el ruido provocado por el tráfico de una carretera. Para las simulaciones se

estudiaron algunas localidades navarras, con el fin de conocer mejor qué factores afectan al nivel de ruido que reciben los habitantes de dichas localidades.

Como complemento a estas simulaciones, se realizó un importante trabajo de campo, con lo que se obtuvieron mediciones y datos reales, lo que permitió comparar los resultados de las simulaciones con los datos obtenidos a pie de carretera.

Una vez comprobada la validez de las simulaciones, se comprobó que el ruido era uno de los principales factores, junto a la contaminación atmosférica, que perjudicaban a los habitantes de localidades atravesadas por carreteras. Del mismo modo, en el presente proyecto han sido estos dos los factores nocivos principales a tener en cuenta, coincidiendo ambos en que sus valores perjudiciales para el ser humano disminuyen en función de la distancia entre la fuente emisora (vehículos) y el receptor.

Partiendo de la hipótesis de que cuanta más distancia separe a las personas de las fuentes emisoras, menor será el daño ocasionado, se ha llegado a la idea de tratar de trazar rutas alternativas que traten de evitar, en la medida de lo posible, el tránsito por zonas pobladas o zonas de especial interés ecológico. La idea es sencilla: cuanto más lejos, mejor.

Aquí se puede ver un ejemplo de cómo se dispersa el ruido provocado por el tráfico de una carretera a su paso por una localidad, en este ejemplo, Elizondo:

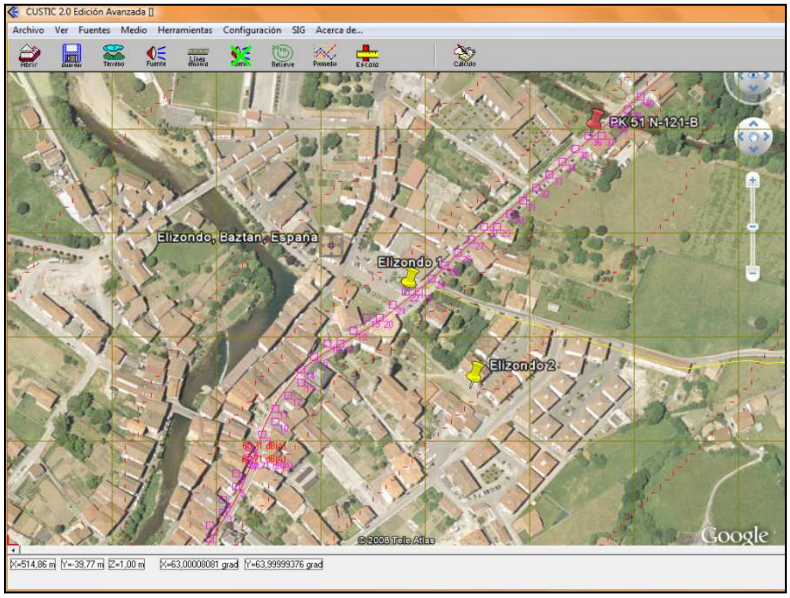


Ilustración 3.1.: Carretera a su paso por ELIZONDO. CUSTIC 2.0. (Elaboración propia)

En esta imagen podemos ver como las isolíneas de nivel acústico se extienden de forma regular a ambos lados de la carretera. Este programa simula la dispersión del ruido para un

supuesto plano, ya que no tiene en cuenta el efecto barrera producido por los edificios y otros obstáculos del terreno.

En el caso de la dispersión del ruido, al contrario que con los gases contaminantes, el viento no juega un papel relevante en su dispersión. El posible efecto que un viento fuerte puede tener sobre el ruido no es más que el provocado por el enmascaramiento del sonido original dentro del ruido del propio viento.

De todos modos, las mediciones realizadas en los propios escenarios simulados, confirman que la expansión del ruido debido al tráfico de las carreteras se produce a ambos lados de la misma con una intensidad similar. A efectos del DSS esta es la información relevante, ya que no va a tener en cuenta las intensidades del ruido generado, pero si la tipología de la población.

En esta otra vista del mismo ejemplo, se puede ver cómo las isolíneas de intensidad acústica se dispersan de forma regular y van disminuyendo conforme se alejan de la línea emisora, en este caso la carretera:

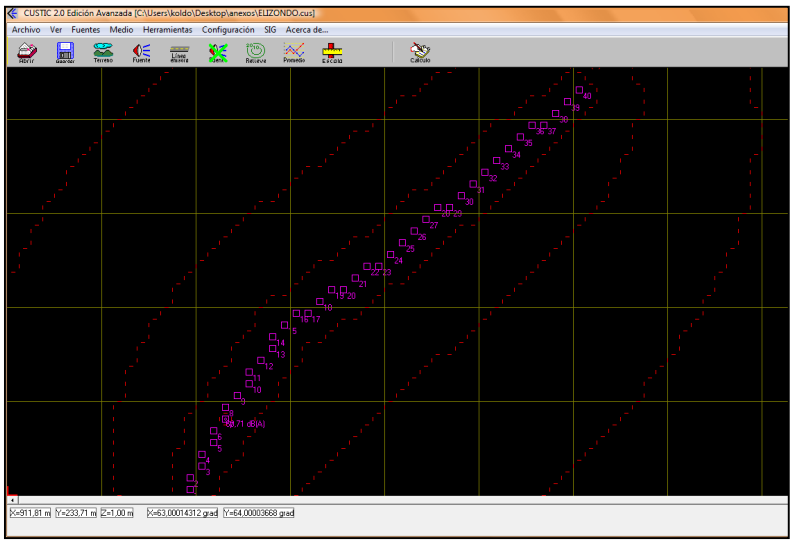


Ilustración 3.2.: Vista de isolíneas acústicas. ELIZONDO. CUSTIC 2.0. (Elaboración propia)

Este programa es capaz de simular la difusión del ruido, ya sea de una fuente emisora puntual o de una fuente lineal, como es el caso de una carretera. En realidad no es más que la superposición de esferas a lo largo del recorrido de la línea.

Si nos fijamos en las anteriores imágenes, se puede ver cómo el ruido generado por el tráfico de la carretera afecta a todo lo que haya a su alrededor, dentro de un rango que dependerá de la intensidad sonora, en el caso de una carretera, depende de la velocidad de tránsito de los vehículos.

En el peor de los casos posibles, como en las imágenes anteriores, cuando la carretera atraviesa el núcleo urbano, el número de personas afectadas es muy elevado, ya que la afección se produce de igual manera a ambos sentidos de la carretera. Desde el punto de vista del DSS, el ejemplo de Elizondo correspondería al tipo de pueblo 3, explicado en detalle en el capítulo 4 de esta memoria.

Por esta razón, uno de los criterios que sigue el DSS es el de buscar rutas que eviten, en la medida de lo posible, el tránsito a través de localidades que no cuenten con variantes, y por tanto, donde sus habitantes se verán más directamente afectados por el ruido del tráfico.

Del mismo modo ocurre con las zonas naturales de especial interés, como pueden ser parques naturales o reservas. En este caso se trata igualmente de reducir los kilómetros totales de nuestras rutas que transitan por dichas zonas naturales.

Aunque no está tan estudiado el impacto del ruido en la flora y la fauna, y resulta más complicado encontrar información al respecto, se sabe que el daño es real, ya que se conoce que afecta a los ciclos naturales de numerosas especies animales. En este proyecto no se pretende entrar en el análisis de cómo afectan los agentes de contaminación en el medio, pero basándome en la información recogida de diferentes agencias de protección de la naturaleza (*WWF, Greenpeace*) y en estudios de impacto ambiental realizados por otros estudiantes de esta universidad, he creído igualmente conveniente dotar al DSS de la posibilidad de trazar rutas que respeten las zonas naturales que sean oportunas en cada caso.

Por otro lado, se buscará minimizar el número de habitantes totales afectados por las rutas seleccionadas. Una consecuencia de esta política es que las poblaciones pequeñas son penalizadas respecto a las grandes, por lo que normalmente serán seleccionadas como puntos intermedios de paso.

Esto puede ser compensado si clasificamos las localidades en función de cómo pasa por ellas el tráfico. Se puede establecer una escala que vaya desde las que cuentan con una variante y por tanto, el impacto del ruido es mínimo, hasta aquellas por las que la carretera atraviesa directamente el centro de la población.

De este modo, las localidades con menor número de habitantes, si bien son perjudicadas por el inferior peso numérico de su población, son favorecidas en el caso de que no cuenten con una variante y sean atravesadas directamente por el tráfico.

3.4.2. Contaminación atmosférica

La contaminación del aire es un serio efecto externo del tráfico generado por el transporte, ya que causa daños que afectan tanto a seres humanos, como a la flora y fauna. En general, los vehículos utilizados en labores de transporte de mercancías por carretera emiten contaminantes como consecuencia de la combustión de combustibles fósiles líquidos, aunque la composición de estas emisiones depende en gran medida de la composición exacta del combustible empleado y de las condiciones en las que se produce dicha combustión.

El transporte, pese a haber disminuido de manera considerable la emisión de algunos contaminantes, ha aumentado su participación en el cómputo de las emisiones totales y es una de las principales fuentes de contaminación, y la más importante en áreas urbanas.

Entre las emisiones contaminantes procedentes del transporte de mercancías por carretera destacan los gases de efecto invernadero, como es el dióxido de carbono (CO_2) proveniente de la combustión completa del carburante. No obstante, no siempre se produce esta operación completamente, por lo que se generan una serie de contaminantes como por ejemplo el monóxido de carbono (CO). Otros contaminantes que provocan este efecto nocivo, comúnmente conocido como cambio climático, son el metano (CH_4) y el óxido nitroso (N_2O). Todos los combustibles contienen además ciertas impurezas como el azufre, que se oxida en la combustión casi en su totalidad en dióxido de azufre (SO_2). Finalmente, a altas temperaturas de combustión el nitrógeno (N_2) se oxida dando lugar a óxido nítrico (NO) y a pequeñas cantidades de dióxido de nitrógeno (NO_2).

Existen ciertas tecnologías como los motores diesel, en las cuales las partículas emitidas se coagulan y sufren otro tipo de procesos que hacen que tengan un espectro doble de tamaños. Estas partículas provienen fundamentalmente de la adición en los combustibles de impurezas como el plomo (Pb), el niquel (Ni), el cobre (Cu), cromo (Cr), zinc (Zn), etc. Además de todos estos contaminantes podemos encontrar otros productos de la combustión como el amoniaco (NH_3), los compuestos orgánicos volátiles (COV) y distintos óxidos de azufre (SO_X) y de nitrógeno (NO_X). Tampoco debemos olvidar que algunos de estos contaminantes primarios, principalmente los NO_X y SO_2, pueden dar lugar a contaminantes secundarios (aerosoles) por transformación química en la atmósfera. Por tanto, las emisiones contaminantes emitidas por un vehículo destinado al transporte por carretera este tipo de transporte son fundamentalmente:

- **PM** (Partículas)
- **NO_x** (óxido de nitrógeno)
- **CO** (monóxido de carbono)
- **CO_2** (anhídrido carbónico)
- **HC** (Hidrocarburos)
- **SO_2** (dióxido de sulfuro)

Los efectos que tienen los contaminantes hasta ahora estudiados sobre la salud y el medioambiente tienen tres niveles: impacto local, regional y global. A diferencia de lo que ocurre con otras fuentes, las emisiones del transporte tienen un impacto más acusado sobre la salud debido a que se producen muy próximas a las personas y a la altura de sus vías respiratorias. Además, dado que las poblaciones se encuentran cada vez más concentradas en las áreas urbanas, los efectos adversos derivados del transporte se agravan debido a esta limitación del espacio. Los costes asociados a los efectos sobre la salud deben estimarse usando estudios epidemiológicos que relacionen la contaminación con las consecuencias negativas para la salud humana.

Los contaminantes emitidos por el transporte también tienen numerosos efectos sobre el medio natural. Algunos ejemplos de ello son la corrosión y ensuciamiento de materiales, eco toxicidad en las cosechas, así como acidificación en los sistemas terrestres y acuáticos. Además, las alteraciones provocadas por estos contaminantes sobre la composición química de la atmósfera derivan en grandes problemas ambientales como la lluvia ácida, la destrucción de la capa de ozono o el cambio climático. El efecto invernadero se debe principalmente a las emisiones de CO_2, NO_X, CH_4 y CO, mientras que la acidificación mayormente por los agentes SO_2 y NO_X.

En lo que se refiere a la evaluación del cambio climático, la estimación de costes es un problema complicado. Por un lado, la elevada incertidumbre sobre los efectos globales a largo plazo se traduce en numerosas hipótesis y escenarios. Por otro lado, se plantea el problema de la reducción de emisiones como responsabilidad política de los países. Por ello, quizá la mejor manera de valorar este impacto sea empleando el método de costes de reparación, basado en las recomendaciones del IPCC para la Unión Europea de reducir a la mitad las emisiones de CO_2 del transporte.

Legislación actual y límites

A continuación se va a resumir la situación de la legislación europea actual con respecto a las emisiones permitidas para vehículos terrestres como automóviles y camiones. Se busca demostrar que, a pesar de los esfuerzos de los estamentos gubernamentales, la tecnología actual en el sector de la automoción sigue siendo altamente contaminante.

No sería justo decir que no han producido avances, ya que la industria automovilística ha invertido e invierte miles de millones de Euros en investigación, pero la realidad es que mientras los motores sigan siendo alimentados por recursos fósiles, inevitablemente se generarán gases y partículas residuales.

El hecho de que un motor de explosión genere estos agentes contaminantes es implícito a su propia naturaleza. Los procesos químicos y físicos a través de los cuales se extrae energía de

los combustibles fósiles, dan como resultado innumerables moléculas de sustancias nocivas para el medio.

Las legislaciones europeas para los motores diesel de los vehículos pesados se recogen tomando como referencia las normas Euro I a Euro V. La norma Euro I para motores medios y pesados se introdujo en 1992, llegando en 1996 la norma correspondiente al Euro II. Estas normas tuvieron su aplicación en vehículos diesel pesados y autobuses urbanos.

En 1999, el Parlamento Europeo y el Consejo de Ministros para el Medio Ambiente adoptaron la norma Euro III (Directiva 1999/96/EC del 13 de Diciembre de 1999), así como las normas Euro VI y Euro V para el periodo 2005/2008. Las normas también dan valores específicos y estrictos para los vehículos de bajo nivel de emisiones o "vehículos ecológicos" con vistas a minimizar la polución atmosférica en las ciudades. En abril del 2001, la Comisión Europea adoptó la Directiva 2001/27/EC que introdujo enmiendas a la Directiva 88/77/EEC que prohibió el uso de mecanismos inservibles de emisiones así como las estrategias irracionales de control de emisiones.

Se espera que los valores límite de las emisiones establecidos para el periodo 2005/2008 requieran nuevos vehículos diesel pesados hechos a medida para dichos niveles de gases de escape. La Tabla 3.3 contiene un resumen de las normas de emisión de contaminantes de los vehículos pesados y sus fechas de implantación en la Unión Europea.

Norma	NO_x (g/Kwh.)	HC (g/Kwh.)	PM (mg/Kwh.)
Euro I (1992-93)	9.0	1.23	400
Euro II (1995-96)	7.0	1.1	150
Euro III (2000)	5.0 [1)]	0.66 [2)]	100/160 [3)]
Euro IV (2005)	3.5 [1)]	0.46 [2)]	20/30 [3)]
Euro V (2008)	2.0 [1)]	0.46 [2)]	20/30 [3)]
Euro V (2008)	*1.0 [1)]*	*0.46 [2)]*	*2/3 [3)]*
Euro VI (2010)	*0.05 [1)]*	*0.46 [2)]*	*2/3 [3)]*
1) Ciclo ESC y ciclo ETC. 2) Ciclo ESC. 3) Ciclos ESC y ETC respectivamente. Nota: También existen valores para CO Y CH_4, pero no se han incluido en esta tabla.			

Tabla 3.3. Normas europeas de emisiones para vehículos pesados, y propuestas UBA para 2008 y 2010

3.4.2.1. Dispersión de los gases contaminantes

Al igual que ocurría con el ruido, los gases contaminantes generados por el transporte siguen un patrón de dispersión en el medio. De los estudios realizados en el proyecto TRANSPIR, se pudo extraer la conclusión de que el nivel de incidencia de estos gases en el individuo se atenúa en función de la distancia a la fuente emisora.

Si en el ruido la dispersión se producía de un modo geométrico, siguiendo la forma de una esfera o de un cilindro, en caso de una línea emisora, en la contaminación atmosférica intervienen otros factores que alteran la forma de dispersión.

Uno de estos factores y, quizás el más importante, es el viento. Los gases contaminantes emitidos por los escapes de los automóviles y camiones tienden a elevarse de forma gradual en la atmósfera, debido a su densidad y a su temperatura. Es durante la elevación donde el viento puede incidir en la dispersión de los gases. Según su intensidad, la "nube" formada por los gases contaminantes es empujada y arrastrada, hasta que los gases se diluyen en el aire de la atmósfera.

Esto hace que la dispersión de los gases sea mucho menos predecible que la del ruido, por lo que elaborar modelos de dispersión resulta más complicado.

Para ello, en el proyecto TRANSPIR se utilizó una herramienta capaz crear modelos de dispersión para gases emitidos por vehículos. Este programa es el DISPER 1.2.

Con esta herramienta informática se estudió el posible comportamiento que los gases contaminantes podían tener en diferentes localidades navarras.

Un ejemplo del modelo de dispersión puede ser el de Irurtzun, a continuación:

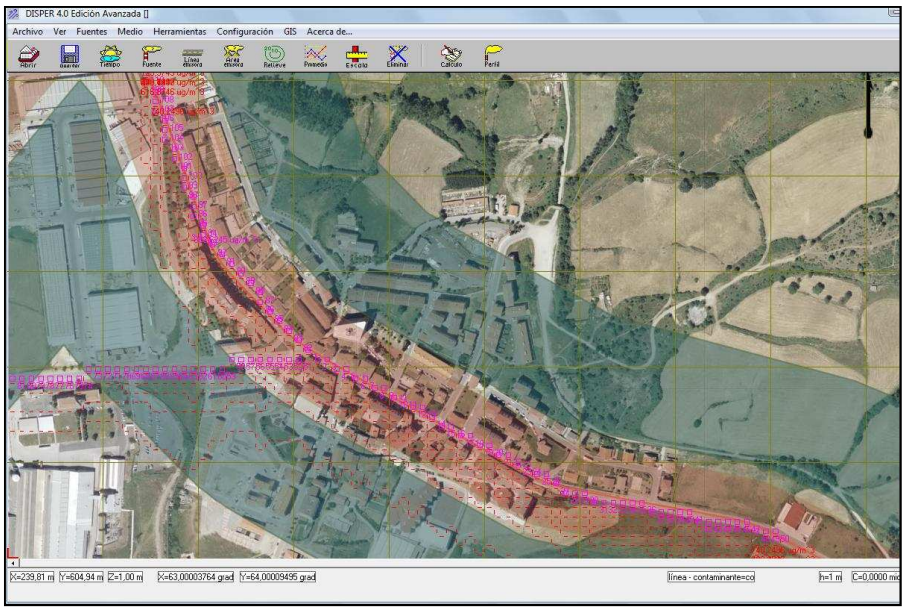


Ilustración 3.3.: Dispersión de CO. Irurtzun. Disper 1.2. (Elaboración propia)

En este caso, el gas estudiado ha sido el CO, teniendo en cuenta los datos de la intensidad de tráfico diario que atraviesa la localidad.

Se aprecia cómo el viento, dirección Norte-Sur, arrastra la "nube" de CO provocada por los vehículos de la carretera. Los datos del viento se tomaron de la información proporcionada por las distintas estaciones meteorológicas del Gobierno de Navarra ubicadas en la zona.

En la siguiente imagen se aprecia mejor la forma de la nube, esta vez sin el fondo del núcleo urbano:

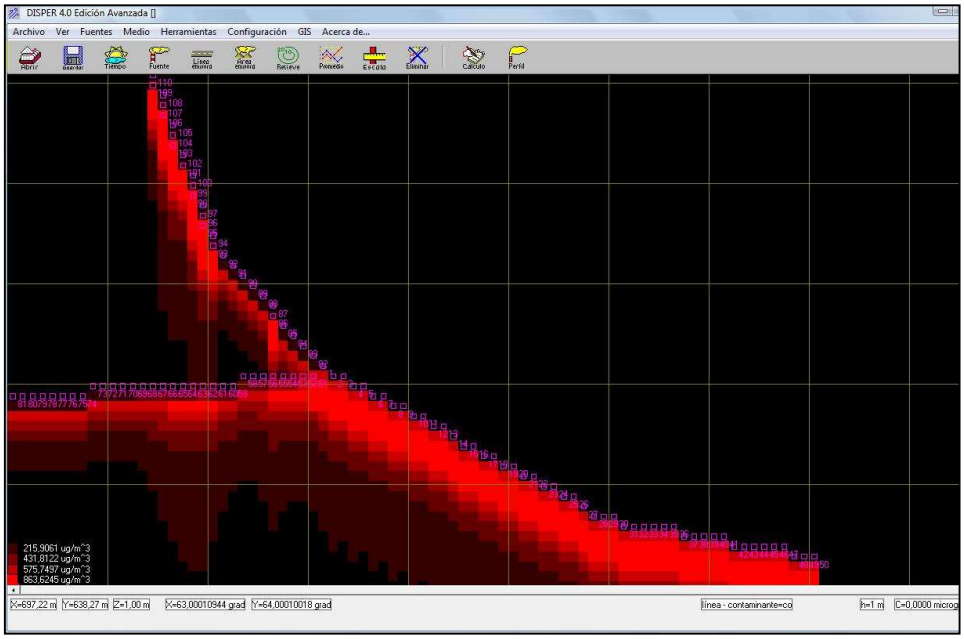


Ilustración 3.4.: Dispersión de CO. Irurtzun. Disper 1.2. (Elaboración propia)

Los siguientes modelos están basados en datos medios de tráfico y de viento dominante en la zona. Sin embargo, observando la información meteorológica disponible, el viento suele ser, por norma general, muy cambiante tanto en dirección como en intensidad.

Lo mismo ocurre con la intensidad de tráfico diario, los datos calculados se basan en medias, pero desglosando la información por horas, podemos ver que el volumen de vehículos cada hora es muy irregular. En horas de la mañana y centrales del día se alcanzan picos de intensidad, y en otros momentos como por la noche o de madrugada apenas existe movimiento de vehículos.

De esta variabilidad de la información se deduce que para casos como el mostrado en las imágenes, el modelo ha de realizarse en múltiples ocasiones, variando los datos de entrada para conseguir abarcar la mayor parte de las posibilidades.

De este modo, la conclusión a la que se llega es que la dispersión atmosférica de los gases, durante las primeras decenas de metros, afectan indistintamente a todo lo que se encuentre a su alrededor, antes de elevarse y diluirse en la atmósfera.

Así pues, al igual que ocurría con el ruido, se puede afirmar que las rutas que eviten poblaciones atravesadas por la carretera y que pasen lo menos posible por núcleos urbanos, serán menos dañinas para la salud y bienestar de las personas.

En el siguiente capítulo, se explican los criterios que se han seguido para clasificar las distintas localidades según su tipo. Así se intenta reflejar la el mayor peso que deben tener aquellas poblaciones de paso en las cuales la carreta atraviesa el centro urbano.

Del mismo modo, las zonas naturales de especial interés pueden ser clasificadas en función de su relevancia, ya sea ecológica, paisajística o por cualquier otro factor del interés del usuario del DSS.

3.5. Conclusión

La conclusión que se extrae del análisis del comportamiento del ruido y de los gases contaminantes provenientes de los vehículos, es que mediante un alejamiento suficiente del tráfico de las personas y de las zonas naturales de especial interés, los efectos nocivos sobre los mismos son mucho menores.

En el caso de la contaminación acústica es más fácil entender esta idea. Cualquier individuo situado a pie de una carretera recibe niveles de ruido superiores a los 80 dB provenientes de los vehículos. A medida que el individuo se aleja de la carretera, la intensidad del ruido que recibe va disminuyendo, hasta que llega un momento en el que el ruido de la carretera se mezcla con el ruido base del ambiente y desaparece.

Si conseguimos alejar nuestras rutas de paso de los núcleos urbanos lograremos disminuir el número de personas afectadas por el ruido de nuestros vehículos. Si esta política la siguieran un número suficiente de empresas y particulares, las reducciones podrían ser significativas.

A diferencia del ruido, los gases contaminantes no se pierden totalmente en la distancia, sino que se diluyen en la atmósfera, acumulándose y contribuyendo a fenómenos como el cambio climático. Estos fenómenos se producen a nivel global, de manera que aunque busquemos rutas que eviten pasar por núcleos de población y por zonas naturales, no podremos evitar por completo los daños causados por los gases de nuestros vehículos.

Sin embargo, los efectos directos de una exposición a altos niveles de estos gases provocan una serie de enfermedades que, en función del tiempo de exposición, pueden ser de leves hasta graves en algunos casos. Es por esta razón por la que sigue siendo la distancia un factor relevante para disminuir todas estas afecciones.

Del mismo modo, en cuanto a las zonas naturales, los niveles altos de gases provenientes de los motores de combustión, afectan a la flora en los procesos de fotosíntesis, ya que los gases como el CO interfieren en la correcta asimilación del Oxígeno y del CO2.

Por estas razones, si bien no conseguimos eliminar todas las afecciones provocadas por el tráfico, mediante la distancia de las rutas a las zonas sensibles, podemos mitigar algunos de los efectos más directos sobre la población y el medio.

Así pues, la idea de recalcular rutas de reparto que disminuyan el tránsito de vehículos por zonas especialmente sensibles, es la base a partir de la cual se diseña el DSS.

4. PRESENTACIÓN DEL GR-DSS

En este capítulo se pretende explicar de forma clara y sencilla las diferentes fases del desarrollo del software que se han seguido durante el diseño y la implementación del DSS. Normalmente, una herramienta software consiste en un programa informático que, dados unos datos de entrada, los analiza, los procesa y finalmente devuelve unos datos de salida.

Estos datos de salida son la solución que el programa ha conseguido tras los diversos procesos de cálculo, muy diferentes en función de la naturaleza del problema. En el caso que nos ocupa, la solución es un conjunto de rutas que cumplen una serie de requisitos inicialmente impuestos.

Para facilitar la explicación del trabajo realizado, así como la comprensión del funcionamiento del programa, he diseñado un ejemplo completo, en el cual existen una serie de clientes dispersos por la geografía navarra, así como unas localidades de especial sensibilidad y unas zonas naturales de interés. Asimismo, se han asignado una serie de valores aleatorios correspondientes a las demandas de los clientes. En cambio, para otro tipo de valores, como las distancias entre dos localidades del problema, se ha tratado de respetar las distancias reales. Lo mismo ocurre con el número de habitantes de cada localidad.

El resto de información, como el tipo de zonas naturales o el tipo de poblaciones de paso son valores subjetivos, asignados a puntos concretos del mapa, buscando que el ejemplo resulte lo más completo posible. Siempre que se ha podido se han asignado valores lo más cercanos a la realidad.

En las páginas siguientes se explicará el funcionamiento de la herramienta software diseñada, y para ello me apoyaré en el anterior ejemplo. De este modo espero conseguir un doble objetivo: Por una parte, que el lector del presente trabajo comprenda más fácilmente los detalles referentes al programa y, por otro lado, validar con un ejemplo los resultados obtenidos con la ejecución del programa.

Así pues, las explicaciones del funcionamiento general de cada parte del programa irán acompañadas por ejemplos extraídos de la ejecución del ejemplo de Navarra.

4.1. Entorno de programación.

El programa ha sido desarrollado en el lenguaje de programación Java. Entre otros motivos, se ha elegido este lenguaje dada su gran facilidad para la programación modular, lo que ha permitido aprovechar el código ya diseñado para el algoritmo SimuRoute y, una vez modificado e integrado, ha servido de núcleo del DSS.

Además, Java es en la actualidad uno de los lenguajes de programación más extendidos y utilizados en infinidad de proyectos informáticos. Por esta razón, varias de las asignaturas de la carrera de Informática de gestión estudian Java y la programación orientada a objetos, por lo que el autor de este proyecto conoce dicho lenguaje.

Para programar en Java se ha utilizado una plataforma de programación muy extendida entre los programadores: NetBeans 6.9.1.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

El **IDE NetBeans** es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso.

Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente.

El NetBeans IDE es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

NetBeans IDE 6.5, la cual fue lanzada el 19 de noviembre de 2008, extiende las características existentes del Java EE (incluyendo Soporte a Persistencia, EJB 3 y JAX-WS). Adicionalmente, el NetBeans Enterprise Pack soporta el desarrollo de Aplicaciones empresariales con Java EE 5, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web servicies (for BPEL), y modelado UML. El NetBeans C/C++ Pack soporta proyectos de C/C++, mientras el PHP Pack, soporta PHP 5.

4.2. Diseño del programa

Para el diseño del programa se siguieron algunas pautas propias de la ingeniería del software, ya que aunque el tamaño del programa no es excesivamente grande, existía una cierta complejidad a la hora de reutilizar el código existente, referente al algoritmo SimuRoute.

Además, se necesitaba un cierto grado de planificación previa, debido a la dificultad de recombinar los métodos y clases previos con unos nuevos, creados con un fin distinto al de los originales.

En muchos casos, la dificultad de la programación basada en un código previo, reside en entender qué cambios se deben hacer y en cómo afectarán al resto del programa.

Por un lado, se realizó un informe con los requisitos de entrada y de salida que tendría el futuro programa, así como del tratamiento que se debía hacer de los datos. El resumen de este informe es el siguiente:

1. Entradas

1.1 Fichero de Nodos a servir:

Id Nombre X Y Demanda Población

1.2 Fichero de reservas naturales

IdOrigen,IdDestino NombreOrigen,NombreDestino Km

TipoReserva X Y

1.3 Fichero de pueblos de paso

IdOrigen,IdDestino NombreOrigen,NombreDestino Nombre

Población Tipo X Y

2. Salidas

2.1 Interfaz

2.1.1 Vista solución

2.1.2 Vista detalle

2.1.2.1 Id Ruta

2.1.2.2 Coste de la ruta (Distancia)

2.1.2.3 Habitantes afectados

2.1.2.4 Coste añadido por Habitantes afectados

2.1.2.5 Km por reserva natural

2.1.2.6 Coste añadido por reserva natural

2.1.2.7 Coste total (suma)

2.2 Fichero salida

2.2.1 Id Ruta

2.2.2 Recorrido (Nombre del pueblo --> Nombre del pueblo)

2.2.3 Id Ruta

2.2.4 Coste de la ruta (Distancia)

2.2.5 Habitantes afectados

2.2.6 Coste añadido por Habitantes afectados

2.2.7 Km por reserva natural

2.2.8 Coste añadido por reserva natural

2.2.9 Coste total (suma)

Informe 4.1. Definición de requisitos de entradas/salidas.

Una vez definidas tanto las entradas como las salidas, el siguiente paso fue definir los procesos y tereas fundamentales que tenía que realizar el programa.

Es lo que se conoce como el flujo de programa:

Flujo de programa

1.Cargar interfaz

1.2 Mensaje de "Seleccione fichero de nodos"
1.3 Valores por defecto de parámetros

2.Cargar nodos

2.1 Evitar mensaje en botón ejecutar
2.2 Mensaje de "Introduzca los parámetros de ejecución"
2.3 Mostrar los nodos en vista solución
2.4 Cargar nodos en estructura lista
2.5 Crear matriz de distancias

3. Cargar Reservas

3.1 Crear matriz de reservas (Cargar fichero en memoria)
3.1.1 Id1 Id2 Km Tipo X Y
3.2 Pintar reservas en vista solución

4. Cargar Pueblos de paso

4.1 Crear matriz de Poblaciones (Cargar fichero en memoria)
4.1.1 Id1 Id2 Hab Tipo X Y
4.2 Pintar pueblos

5. Ejecutar

5.1 Si no existe fichero de nodos pedirlo
5.2 Leer parámetros
5.3 Calcular matriz de costes a partir de las 3 matrices
5.4 Pasar matriz de costes junto con parámetros
5.5 Ejecución SimuRoute
5.6 PostProceso de las soluciones
5.6.1 Recorrer las aristas y mirar
5.6.1.1 Pob C.Pob KM Reserv C.Reserv
5.7 Pintar solución
5.8 Mostrar detalles de salida
5.9 Crear ficheros de salida

Informe 4.2. Flujo básico del programa.

A continuación, se planificaron las tareas principales a realizar para el correcto desarrollo del programa. Esta planificación sirvió de guía a la hora de programar, ayudando a resolver errores y permitiendo una buena distribución del tiempo disponible.

1. Planificación de tareas a realizar

1.1 Diseñar interfaz

1.1.1 Vista de rutas

1.1.1.1 Buscar imagen de fondo

1.1.1.2 Ajustar Nodos a imagen

1.1.1.3 Ajustar Nombres en la imagen

1.1.1.4 Opción de mostrar mapa de fondo

1.1.2 Parámetros de ejecución

1.1.2.1 Cargar nodos necesario(Muestra los nodos en la vista de rutas)

1.1.2.2Posibilidad de cargar reservas naturales (Pintamos óvalo verde)

1.1.2.3 Posibilidad de cargar poblaciones (Pintamos circulo azul)

1.1.2.4 Capacidad del vehículo

1.1.2.5 Longitud máxima de la ruta

1.1.2.6 Ajuste de reserva natural

1.1.2.6 Ajuste de población

1.1.3 Vista detalle de solución

1.1.3.1 Id Ruta

1.1.3.2 Coste de la ruta (Distancia)

1.1.3.3 Habitantes afectados

1.1.3.4 Coste añadido por Habitantes afectados

1.1.3.5 Km por reserva natural

1.1.3.6 Coste añadido por reserva natural

1.1.3.7 Coste total (suma)

1.1.4 Fichero de salida

1.1.4.1 Id Ruta

1.1.4.2 Recorrido (Nombre del pueblo --> Nombre del pueblo)

1.1.4.3 Id Ruta

1.1.4.4 Coste de la ruta (Distancia)

1.1.4.5 Habitantes afectados

1.1.4.6 Coste añadido por Habitantes afectados

1.1.4.7 Km por reserva natural

1.1.4.8 Coste añadido por reserva natural

1.1.4.9 Coste total (suma)

1.2 Escribir Fichero de Nodos a servir

1.3 Escribir Fichero de reservas naturales

1.4 Escribir Fichero de pueblos de paso

1.4.1 Arreglar coordenadas para quede en la arista

1.5 Leer y comprobar que lee bien los ficheros

1.5.1 Fichero de Nodos a servir

1.5.2 Fichero de reservas naturales

1.5.3 Fichero de pueblos de paso

1.6 Construir matriz de costes

1.6.1 Matriz de distancias

1.6.2 Matriz de distancias + reservas

1.6.3 Matriz de distancias + reservas + poblaciones

1.7 Enlazar con el SimuRoute

3.7.1 Modificar para que la matriz de coste sea un parámetro

1.8 Obtener las salidas

Informe 4.3. Planificación de tareas.

Gracias a estos informes previos, el proceso de programación fue mucho más sencillo. El conocer exactamente qué información se quería obtener desde un principio facilitó el desarrollo del programa.

La dificultad mayor se encontró a la hora de combinar y reutilizar el código del SimuRoute, ya que hubo que adaptarlo a unos nuevos requisitos de entradas y salidas.

4.3. Funcionamiento del programa

A continuación, se explica el modo de trabajar del DSS desde el punto de vista del programador, pero también de una forma clara, que pretende ayudar a la comprensión del programa por parte de todo tipo de usuario.

Para ello se mostrarán distintas partes del código, las más relevantes, así como ejemplos sencillos allí donde proceda.

Al tratarse de una herramienta en la que las soluciones gráficas tienen una gran importancia, en muchos casos se muestran imágenes de la interfaz, con el fin de ayudar al lector a la comprensión del problema.

4.3.1 Entradas

Una buena forma de comenzar a entender el funcionamiento del DSS es explicando detenidamente cuales son los datos necesarios para su correcta ejecución.

Ya se ha hablado en el capítulo 1 del VRP, así como de los diferentes métodos algorítmicos que existen para encontrar buenas soluciones. Al tratarse de un problema en el que hay que calcular rutas para recorrer todos los nodos de un grafo, es evidente que previamente deberemos conocer la información relativa a dichos nodos. Así pues, deberemos proporcionar al programa los datos que definan correctamente el grafo, para que a partir de ahí, él pueda calcular las mejores rutas.

Además de buenas rutas, este DSS pretende encontrar aquellas que sean menos perjudiciales para las personas y para el medio ambiente. Por lo tanto, al igual que con los nodos, deberemos facilitar al programa la información referente a aquellos lugares que nos interesa evitar en nuestras rutas.

Toda esta información corresponde a lo que podemos llamar los datos propios del problema, ya que en sí mismos son la definición del problema a resolver.

Al trabajar internamente con un algoritmo (SimuRoute, Capítulo 1) de cálculo de rutas, el DSS necesita ajustar una serie de valores internos con los que trabaja el algoritmo. A estos valores los llamaremos parámetros de entrada.

El DSS proporciona una forma fácil y rápida de ajustar estos parámetros, a través de un menú desplegable de la interfaz gráfica. Esto da al usuario la posibilidad de recalcular tantas veces como quiera el mismo problema cambiando alguno de estos parámetros, hasta encontrar la solución más conveniente a sus propósitos.

A continuación se detalla en qué consisten todos estos datos de entrada, tanto los datos del problema como los parámetros de ejecución, y cómo hemos de estructurarlos para que el DSS pueda trabajar a partir de ellos.

4.3.1.1. Parámetros de ejecución

Los parámetros de ejecución han sido separados en dos grupos. Por una parte están los más básicos y susceptibles de ser cambiados por el usuario.

En este grupo se encuentran:

1. La carga del vehículo
2. La longitud Máxima de la ruta
3. Ajuste de poblaciones
4. Ajuste de zonas naturales

Se encuentran en la parte derecha de la pantalla principal de la interfaz gráfica:

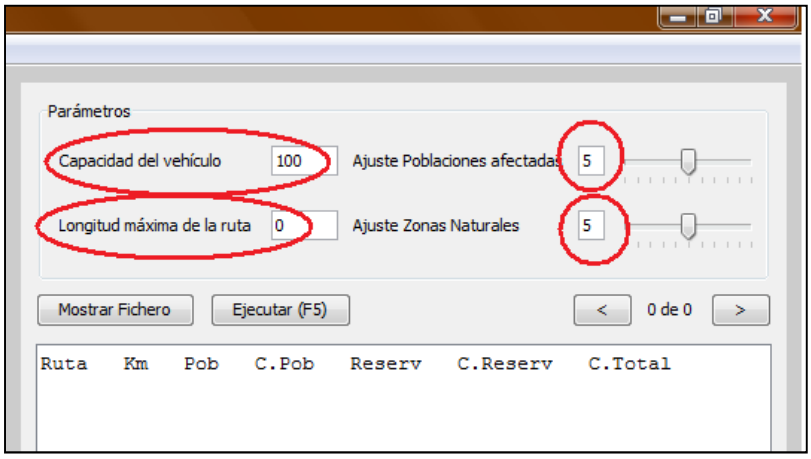


Ilustración 4.8. Parámetros de ejecución. GR-DSS

Por otro lado, el algoritmo necesita otra serie de parámetros de ejecución, más técnicos, y que el usuario normalmente no necesitará cambiar. Con ellos podemos ajustar los cálculos de las rutas para acercarnos lo más posible al óptimo.

Se encuentran en una ventana emergente aparte, que se despliega desde el menú:

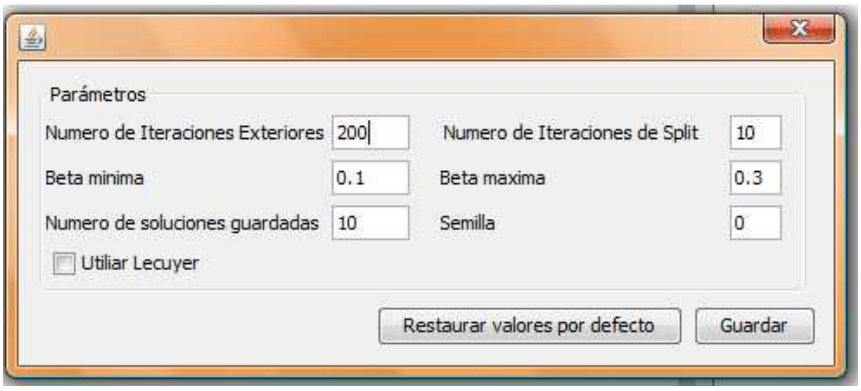


Ilustración 4.9. Parámetros de ejecución (2). GR-DSS

4.3.1.2. Ficheros de entrada

Para la entrada de datos, se ha decidido un método que viene siendo utilizado por los investigadores para probar la efectividad y eficacia de sus algoritmos para el VRP. El método consiste en crear ficheros de texto plano, que contengan la información de cada cliente así como la del almacén.

La información que normalmente se necesita para representar a los clientes no es más que su demanda y sus coordenadas cartesianas. Para su simplificación, los algoritmos que se diseñan para este tipo de problemas trabajan con coordenadas X e Y de modo que se representan los clientes como puntos en el plano. En nuestro caso, el algoritmo SimuRoute en el que se basa el DSS trabaja de esta manera, y a partir de esta información él calcula la matriz de costes con las distancias que existen entre cada par de puntos.

Este método permite que la información necesaria de cada cliente sea la mínima, puesto que el objetivo principal de los investigadores del VRP suele ser el optimizar el algoritmo, ya sea tratando de reducir los costes de cálculo o buscando las soluciones más cercanas a la mejor conocida.

La sencillez del método de entrada de información permite diseñar numerosas pruebas de forma rápida y sencilla, pero también tenemos algunas limitaciones que vienen impuestas por esta forma de trabajar. Las principales limitaciones son:

a) **Líneas rectas:** El algoritmo calcula automáticamente las distancias entre cada par de puntos correspondientes a cada cliente. Esta distancia se denomina distancia euclídea entre dos puntos del plano, los cuales se pueden representar por $A(x_1, y_1)$ y $B(x_2, y_2)$. No es más que la longitud del segmento de recta que tiene por extremos A y B. Puede calcularse así:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Como es evidente, esta distancia no corresponde a la distancia real que debemos recorrer para llegar de A hasta B en la práctica. Los trazados de las carreteras se diseñan conforme a la orografía y el relieve, por lo que continuamente encontramos curvas y desniveles.

El resultado es que la distancia real entre dos puntos siempre será mayor que la que el algoritmo ha estimado. Sin embargo, al producirse el mismo error con todos los pares de puntos, el fallo relativo se minimiza y la simulación aumenta su validez.

Para el futuro sería interesante contar con información de las distancias reales entre cada par de puntos, lo que haría que los cálculos realizados por el algoritmo fueran más exactos. Esto se explica en el capítulo 6, Mejoras futuras.

b) **Todos los puntos conectados:** El otro gran problema de esta forma de tratar la información de entrada es que el algoritmo da por válidas todas las conexiones entre cada par de puntos, es decir, desde cada punto podemos viajar hasta cualquier otro directamente.

Sabemos que en la realidad esto no es así. No existe una carretera para ir de cada población a cualquier otra directamente.

Este problema puede ser solucionado fácilmente mediante el DSS diseñado en este proyecto. La solución se detalla en el capítulo 6, Mejoras futuras, pero consistiría en asignar grandes costes a las rectas que no se corresponden con una carretera en la realidad.

Para el almacén, la información requerida es mínima, ya que sólo es necesario conocer sus coordenadas en el plano y su demanda será siempre 0. En este proyecto no se estudia el problema del stock disponible, se supone que la cantidad de mercancías en el almacén es ilimitada.

Hasta aquí se ha explicado cómo están estructurados los ficheros de entrada de los problemas VRP tradicionales. A continuación se va a explicar en detalle las modificaciones realizadas a este formato de entrada, así como una vista en detalle de cómo se ha elaborado el ejemplo aplicado a un supuesto VRP en Navarra.

Además, el DSS tiene que trabajar con información referente a las zonas naturales y a las poblaciones sensibles para tratar de diseñar rutas que las eviten. Esta información se ha estructurado siguiendo el mismo formato que la información de los clientes. Se han creado dos ficheros, cada uno de los cuales contiene la información necesaria para identificar, localizar y tratar de evitar el tráfico en zonas de especial sensibilidad.

Para leer la información de los ficheros de manera correcta, se ha diseñado un método que permite trocear las líneas de un fichero de texto. En este caso se ha hecho de forma que los diferentes valores estén separados por tabuladores o por espacios en blanco, no importando la cantidad de ambos entre valores.

El método utilizado para trocear las cadenas pertenece a la clase DSSInOut y es el siguiente:

```
    // Funcion que trocea la linea e ignora tabuladores de más.
    private static String[] mySplit(String line) {
        String[] OldLine = line.split("\t");
        String[] myLine = new String[OldLine.length];
        int i = 0;
        for (String token : OldLine) {
            if (!token.equals("")) {
                myLine[i] = token;
                i++;
            }
        }
        return myLine;
    }
}
```

Código 4.1. Método mySplit. GR-DSS

El método anterior es utilizado por los diferentes métodos encargados de extraer la información de los ficheros de texto y almacenarla de forma correcta en la memoria.

Un ejemplo de dichos métodos puede ser el *getNodesList,* que una vez seleccionado la ruta y el nombre del fichero de nodos, crea un ArrayList de estructuras “DSSNode”. Cada uno de estos DSSNode contiene la información disponible de cada nodo, es decir, su ID, su nombre, sus coordenadas X, Y, su demanda y su población.

Por último, el método concluye devolviendo el ArrayList "nodeList" que contiene toda la información de todos los nodos del fichero de entrada.

```
// Lee y guarda los Nodos del fichero de nodos
public static ArrayList<DSSNode> getNodesList(String inputNodesFileName) {
    ArrayList<DSSNode> nodesList = new ArrayList<DSSNode>();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(inputNodesFileName));
        String line;
        while ((line = reader.readLine()) != null) {
            String[] values = mySplit(line);

            int Id = Integer.valueOf(values[0]);
            String Nombre = values[1];
            int X = Integer.valueOf(values[2]);
            int Y = Integer.valueOf(values[3]);
            int Demanda = Integer.valueOf(values[4]);
            int Poblacion = Integer.valueOf(values[5]);

            DSSNode aNode = new DSSNode(Id, Nombre, X, Y, Demanda, Poblacion);
            nodesList.add(aNode);
        }
    } catch (IOException exception) {
        System.out.println("Error processing input nodes file: " + exception);
    }
    return nodesList;
}
```

Código 4.2. Método getNodesList. GR-DSS

A continuación se explican detalladamente los tres ficheros de entrada necesarios para el funcionamiento del DSS:

1. **El fichero de nodos**
2. **El fichero de poblaciones de paso**
3. **El fichero de zonas naturales**

Como parte de la explicación de cada uno de estos ficheros, se adjunta el desglose de los ficheros utilizados en la elaboración del ejemplo del supuesto de Navarra.

1. Fichero de nodos

Este fichero contiene las coordenadas cartesianas correspondientes a cada cliente, así como sus demandas de mercancías. Para esta simulación, las coordenadas representan los Km de distancia respecto al origen (el almacén). Estos valores pueden adaptarse a otras necesidades, como pueden ser el cambio a metros, a millas… o simplemente, afinar más las distancias introduciendo decimales.

La demanda se expresa como un número entero positivo, o 0 en caso de ser el almacén. Estos números no representan ninguna magnitud en concreto, son solo una forma abstracta de simular las peticiones de los clientes. Para la aplicación del DSS a un caso real, se deberían

ajustar estos valores de la forma más conveniente, ya sea representando Kg de mercancía, números de cajas o piezas, volúmenes…

Además de esta información, se ha añadido una columna con un identificador numérico para cada cliente, así como el nombre de la localidad correspondiente. Este identificador facilita la labor de programación, además de permitir identificar pedidos distintos en una misma localización.

Otro dato de interés para el DSS es conocer el número de habitantes de las localidades de los clientes, para, de este modo, calcular un número aproximado de posibles personas "afectadas" por la ruta de reparto.

Esta es toda la información necesaria para poder crear un fichero de nodos clientes. En el supuesto diseñado en este proyecto, el fichero de nodos contiene la siguiente información:

ID	Nombre	Coordenada X	Coordenada Y	Demanda	Número de habitantes
0	Pamplona	0	0	0	198000
1	Elizondo	10	39	31	3050
2	Irurtzun	-18	13	9	2000
3	Noain	2	-5	21	6800
4	Burguete	25	20	25	319
5	Gorriti	-25	28	7	147
6	Irurita	8	36	4	850
7	Olague	4	17	9	203
8	Ostiz	4	11	22	91
9	Sunbilla	-2	40	13	659
10	Zubiri	12	13	10	1547
11	Berrioplano	-4	5	25	582
12	Bera	-5	53	25	3691
13	Liedena	30	-21	14	329
14	Espinal	22	18	7	249
15	Alsasua	-43	12	25	7623
16	Estella	-32	-14	34	14200
17	Tafalla	-7	-31	24	11400
18	Puente_la_Reina	-14	-14	9	2600
19	Peralta	-11	-52	12	5368
20	Los_arcos	-45	-26	10	1372
21	Tudela	3	-82	27	36000
22	Carcastillo	16	-46	15	2670
23	Sanguesa	29	-25	20	5200
24	Lodosa	-35	-41	6	4776

Tabla 4.2. Fichero de nodos cliente

La anterior tabla representa un problema VRP en el cual el almacén está situado en las coordenadas (0,0). Existen 24 clientes distribuidos por toda la geografía navarra, los cuales tienen distintas demandas de mercancías por satisfacer.

La disposición geográfica de los clientes y del almacén, vistos en el mapa sería la siguiente:

Ilustración 4.10. Ubicación de los nodos en el mapa de Navarra.

Se ha procurado elegir poblaciones dispersas por toda la geografía navarra, si bien existe una mayor concentración de localidades en las vías que van hacia el norte. El motivo es que estas vías son las principales receptoras del flujo de tráfico que cruza la frontera hacia Francia a través de la Comunidad Foral de Navarra y por tanto, fueron el centro del estudio realizado en el proyecto TRANSPIR.

En el apartado del manual de funcionamiento se detalla cómo se han de cargar los ficheros de entrada, pero aquí tenemos una vista de cómo quedan ubicados los nodos en el panel gráfico de la interfaz del DSS.

El panel está diseñado para que sea capaz de escalar y centrar cualquier problema que se le introduzca, de manera que sea capaz de aprovechar el máximo espacio para dibujar las localidades de los clientes.

Dichas localidades son representadas con una pequeña imagen y con su nombre a la derecha.

El escalado permite mantener las proporciones reales que existen entre los nodos, consiguiéndose así un dibujo bastante fiel a la realidad. Además, el diseño seguido permite que al redimensionar la ventana de la interfaz, se calcule automáticamente la escala y se redibujen los nodos adaptándose a las nuevas dimensiones del panel. En todo caso, se recomienda trabajar con la ventana maximizada para poder apreciar mejor el dibujo.

La carga de este fichero en el DSS da como resultado la siguiente imagen:

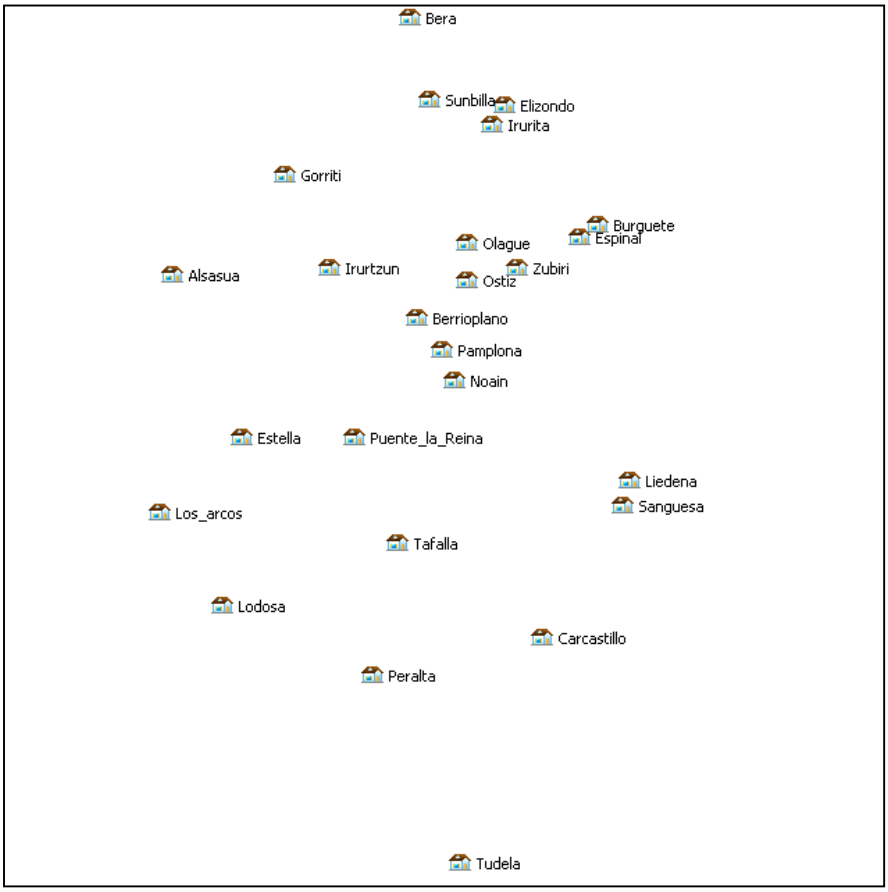


Ilustración 4.11. Ubicación de los nodos. Vista del GR-DSS

El programa calcula la escala a utilizar en función de la altura o de la anchura de cada caso concreto. Elegirá el menor de estos dos valores para el escalado, de esta forma se consigue que todos los nodos puedan ser representados dentro del panel. Al escalar sólo en función de

una de las dos dimensiones, conseguimos mantener una imagen proporcionada y más fiel a la realidad, aunque perjudica al aprovechamiento del espacio total del panel.

Requisitos del fichero de nodos:

Para poder diseñar nuevos problemas y hacerlos correr en el DSS, se han de tener en cuenta una serie de requisitos para mantener el formato de entrada:

- El ID 0 corresponde siempre al almacén.
- Los nombres de las localidades han de formar una única cadena de caracteres. Es decir, los nombres de localidades formados por más de una palabra han de seguir el siguiente formato: Puente_la_Reina.
- Entre columnas de valores pueden introducirse tantos espacios en blanco o tabuladores como se quiera.
- Hay que respetar el orden de las columnas, en caso de introducir los valores desordenados, el programa falseará los resultados o simplemente fallará al intentar cargar el fichero.
- Tener cuidado en no dejar líneas intermedias vacías ni tampoco al final del fichero.
- Guardar el fichero con extensión ".txt"

2. Fichero de poblaciones de paso

El fichero de poblaciones de paso representa aquellas poblaciones que, sin estar incluidas en problema por no pertenecer a ningún cliente, y por tanto no ser objetivos de las rutas, tienen una especial relevancia y se pretende que las rutas solución las eviten.

Esto sucede cuando es conocido que para llegar de A hasta B de un modo directo, es inevitable hacerlo a través de C.

En este caso, tanto A como B pertenecen al problema, ya que hay que hacer repartos en esas poblaciones, pero la localidad C, que no tiene ninguna demanda, sin embargo se ve afectada por el tráfico necesario para ir de A hasta B.

Como se explica en el capítulo 3, en el apartado de externalidades, la localidad C está sufriendo un perjuicio para que las localidades A y B se favorezcan de un reparto comercial. Existen casos en los que estas situaciones pueden ser extremas, como en el caso de que la carretera que pasa por C sea una travesía urbana y atraviese el centro de la localidad.

Lo que se pretende con este fichero es localizar aquellas localidades del problema que sufren los perjuicios del transporte sin ser objetivos de reparto y tratar de sacarlas de las rutas solución. De esta manera se minimizan el número de personas afectadas directamente por nuestras rutas, tratando de que sólo los habitantes de las localidades de reparto sean los afectados. El daño a estos habitantes no puede disminuirse desgraciadamente mediante el rediseño de las rutas, por lo que se darán por buenas las soluciones que impliquen al mínimo número de habitantes afectados.

Las posibles soluciones que nos brinda la opción de incluir poblaciones de paso son muchas, pero los casos más importantes que se han identificado son dos.

El primero se trata de una situación en la que hay involucrados dos clientes A y C con demandas de mercancías. Para llegar de A hasta C existen 2 posibles caminos: El más corto atravesando una localidad de paso B, cuya demanda es 0. El otro camino, más largo, atravesando una localidad de paso D.

El algoritmo del DSS, en función de los ajustes de corrección aplicados, valorará y elegirá cuál de las dos opciones es menos “costosa”. Ya se ha explicado que el coste es una suma de las distancias entre dos puntos más los “daños” ocasionados por el camino (poblaciones de paso y zonas naturales).

Dependerá de los pesos que demos a factores como el número de habitantes o el tipo de localidad (si es atravesada por la carretera, solo rodeada…) y el algoritmo elegirá aquella opción que le lleve de un cliente a otro de la forma menos costosa.

Una ilustración gráfica de esta posible situación, donde para ir de A hasta C lo podemos hacer a través de B o bien a través de D:

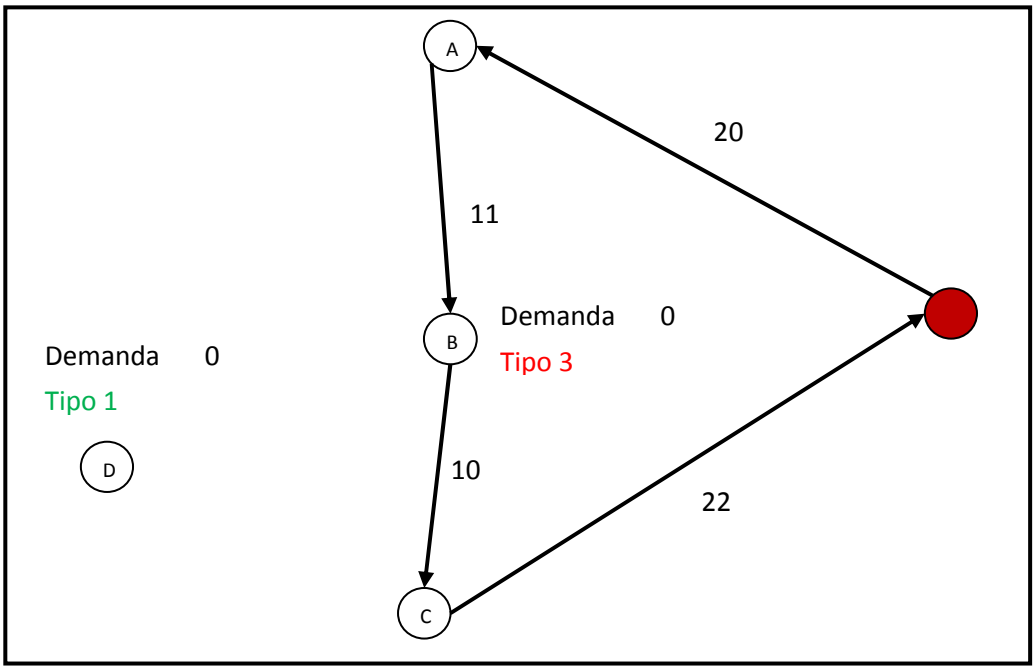


Ilustración 4.12. Elaboración propia.

En esta ilustración, el almacén está representado por el círculo rojo. Los pueblos de paso son B y D, ya que ellos no tienen demandas. A y C son localidades de clientes con demandas conocidas.

Como se puede ver, el camino más directo es a través de B, acumulando un coste kilométrico de 63 Km.

Según el diseño del DSS, el coste total de ir de A hasta C atravesando B, será igual a:

CosteA_B_C = 63 + (Número de habitantes * Ajuste_Tipo * Ajuste_de_corrección)

Este coste será almacenado internamente en la matriz de costes, a partir de la cual, el algoritmo seguirá buscando alternativas mejores.

En el caso de encontrar un camino alternativo cuyo coste total sea menor, el algoritmo lo sustituye en la matriz de costes y vuelve a empezar la búsqueda de rutas mejores. Finalmente, la ruta solución es aquella que se forma con los valores mínimos encontrados en dicha matriz de costes.

En la siguiente ilustración se muestra la otra posible opción:

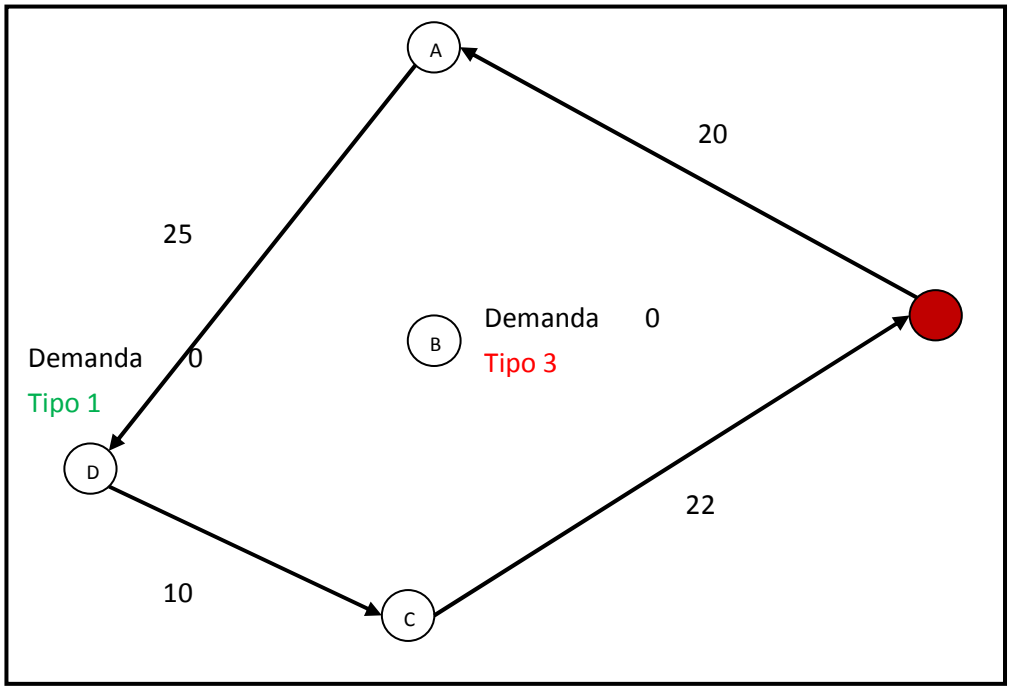


Ilustración 4.13. Elaboración propia.

En este otro caso, el camino elegido para llegar de A hasta C es atravesando D. El coste kilométrico de esta opción es de 77 Km, es decir, 14 más que la primera opción.

No obstante, el DSS precisamente está diseñado para valorar los costes producidos por los perjuicios a la población además de las distancias. En este caso, el coste de esta ruta es:

CosteA_D_C = 77 + (Número de habitantes * Ajuste_Tipo * Ajuste_de_corrección)

El DSS hace estos cálculos y de esta forma elige como la mejor ruta aquélla en la cual el coste total es menor.

Se puede ver que el resultado de las rutas seleccionadas depende en gran medida de los valores de corrección y de tipo que se configuren en el programa.

En cuanto a los valores de ajuste, el ajuste de corrección no es más que un valor que el usuario puede cambiar fácilmente en tiempo de ejecución desde la pantalla principal de la interfaz del programa:

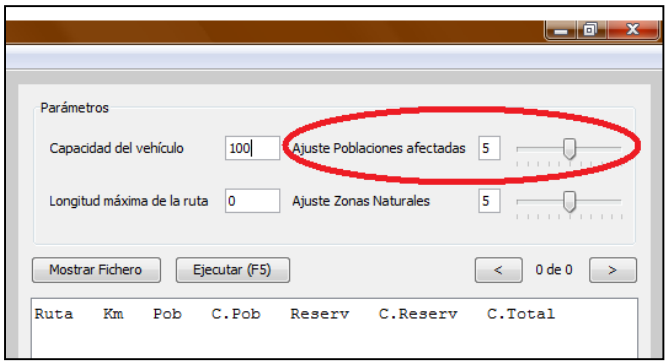


Ilustración 4.14. Valores de ajuste. GR-DSS

De esta manera, podemos ir cambiando la relevancia que le damos a los distintos ficheros cargados en el mapa. Dando el valor 0, el programa trabajará como si no existieran pueblos de paso en el problema, y por tanto, veremos cómo las rutas generadas probablemente atraviesen dichos pueblos.

A medida que aumentamos el valor del Ajuste de pueblos de paso, las nuevas rutas tendrán cada vez mayor facilidad para evitar los pueblos de paso. Esto permite al usuario obtener una gran cantidad de soluciones diferentes con un tiempo mínimo, ya que no es necesario cargar ningún fichero de nuevo, basta con pulsar el botón de Ejecutar y se generará una nueva solución con los valores de ajuste seleccionados.

En el ejemplo de Navarra se han seleccionado 4 poblaciones clasificadas como de paso. Estas 4 poblaciones afectan a cinco posibles rutas entre distintas localidades de clientes. Para este ejemplo no se han tenido en cuenta si existen problemas derivados del tráfico reales en dichas poblaciones de paso, si no que se han buscado aquellas localidades que por su situación geográfica y por estar entre dos nodos clientes, permiten ver de forma clara los cambios que se producen en las rutas solución.

Dicho esto, el fichero de entrada de localidades de paso en el ejemplo de Navarra contiene la siguiente información:

ID Origen	ID Destino	Nombre Origen	Nombre Destino	Nombre	Habitantes	Tipo	C. X	C. Y
19	24	Peralta	Lodosa	Andosilla	1000	3	-26	-48
2	5	Irurtzun	Gorriti	Lekumberri	850	2	-22	21
0	4	Pamplona	Burguete	Aoiz	2000	2	13	9
0	14	Pamplona	Espinal	Aoiz	2000	2	13	9
18	19	Puente_la_Reina	Peralta	Larraga	1000	3	-13	-24

Tabla 4.3. Fichero de poblaciones de paso

Como se puede ver en la tabla, el fichero contiene, además del nombre del pueblo de paso y sus coordenadas en el plano, los datos referentes a los nodos clientes que se ven afectados por el pueblo de paso.

Es necesario proporcionar esta información para que el programa sea capaz de identificar que aristas de la ruta van a tener que ser recalculadas, añadiéndoles al coste kilométrico original el coste añadido por atravesar una población de paso.

Además de estos datos, es necesario conocer el número de habitantes del pueblo de paso, ya que este será un factor que intervenga a la hora de recalcular el coste de la arista en cuestión.

Se incluye la posibilidad de indicar, como una escala de valores, la tipología de la población de paso. En este caso se han descrito tres supuestos diferenciados, lo cual no quita para que en otros supuestos se identifiquen un número distinto de tipologías. Los tres tipos son:

- **Tipo 1:** La carretera no atraviesa de ningún modo la zona habitada de la localidad. Es el tipo menos dañino pero aún así se ha querido tener en cuenta. Basándome en el conocimiento adquirido durante mi participación en el proyecto TRANSPIR, los efectos nocivos directos del tráfico por carretera se sienten incluso a varios cientos de metros, si bien su intensidad decrece en función de la distancia.
- **Tipo 2:** La carretera "roza" o atraviesa de forma parcial el núcleo residencial de la localidad. En este caso, al menos una parte de la población se ve afectada directamente por el ruido y la contaminación de los vehículos que transitan la localidad.
- **Tipo 3:** la carretera atraviesa directamente el centro de la localidad. Este caso es muy claro en algunas localidades de la ribera navarra, donde parece que las viviendas se han ido edificando siguiendo la carretera a ambos lados. También encontramos casos de este tipo en otras zonas de Navarra, como pudiera ser el caso de Irurtzun.

  Es el peor caso de los tres, ya que la mayoría de los habitantes de la localidad están expuestos de forma directa a los gases y el ruido emitidos por los vehículos. Es el caso que más se pretende evitar a la hora de diseñar rutas con el DSS.

Durante la ejecución del programa, el fichero de nodos cliente ha de estar cargado previamente para poder cargar el fichero de poblaciones de paso. Esto se debe principalmente a que, para representar el problema de manera gráfica en el panel, se necesitan los procesos de escalado y centrado del conjunto de nodos, y una vez tomada estas referencias, el programa ya puede dibujar las poblaciones de paso.

Las anteriores poblaciones de paso quedan ubicadas de la siguiente manera en el mapa:

Ilustración 4.15. Distribución de las poblaciones de paso.

En la elaboración del ejemplo del supuesto de Navarra, se han escogido un número pequeño de poblaciones de paso, así como de zonas naturales de interés, con el fin de que las soluciones obtenidas sean sencillas y fáciles de comprender. Evidentemente, el DSS permite la entrada de todos los obstáculos que sean necesarios, repercutiendo únicamente en el tiempo de cálculo y en el uso de la memoria.

Con estos cuatro pueblos de paso, que repercuten en el coste total de cinco posibles aristas, es suficiente para ilustrar el comportamiento de las rutas calculadas. Durante la ejecución del programa, el usuario puede comprobar cómo las rutas tratan de esquivar los pueblos de paso, siempre en función de los valores de corrección y los ajustes seleccionados.

Este es el resultado de cargar el fichero de pueblos de paso en el programa:

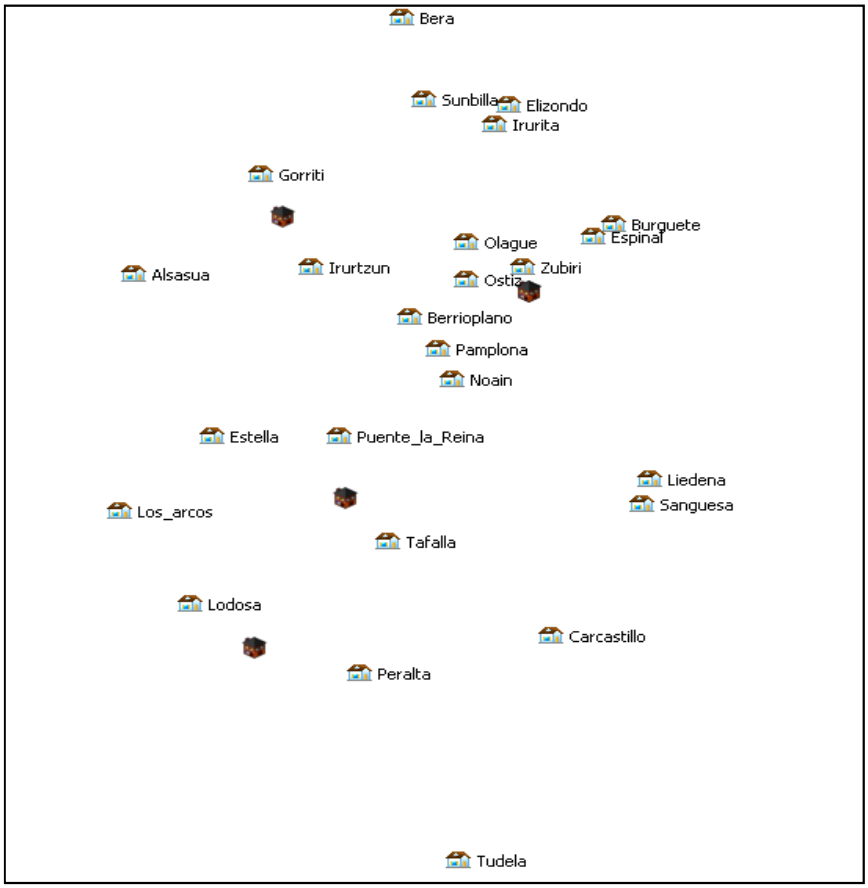


Ilustración 4.16. Distribución de las poblaciones de paso. Visor del GR-DSS

En este caso, los pueblos de paso son representados por una imagen distinta a las poblaciones correspondientes a los nodos cliente, para su fácil localización.

Las localidades de este ejemplo han sido elegidas con el fin de que la ejecución del programa con distintos valores de ajuste y parámetros sea ilustrativa y muestre claramente los objetivos que se pretenden conseguir.

Finalmente, las cinco aristas afectadas por estas cuatro poblaciones son:

a. *Lodosa—Peralta (Andosilla)*
b. *Peralta —Puente la Reina (Larraga)*
c. *Pamplona—Burguete (Aoiz)*
d. *Pamplona—Espinal (Aoiz)*
e. *Irurtzun—Gorriti (Lekumberri)*

Es en estos tramos donde deberemos prestar una mayor atención para apreciar los mayores cambios en las rutas generadas.

Requisitos del fichero de pueblos de paso:

- Los nombres de las localidades han de formar una única cadena de caracteres. Es decir, los nombres de localidades formados por más de una palabra han de seguir el siguiente formato: Puente_la_Reina.
- Entre columnas de valores pueden introducirse tantos espacios en blanco o tabuladores como se quiera.
- Hay que respetar el orden de las columnas, en caso de introducir los valores desordenados, el programa falseará los resultados o simplemente fallará al intentar cargar el fichero.
- Tener cuidado en no dejar líneas intermedias vacías ni tampoco al final del fichero.
- El orden del ID_Origen y el del ID_Destino han de ser de menor a mayor. Con los dos extremos se describe la arista que los une. Por motivos de programación, es necesario introducir de este modo las aristas, el nodo con ID menor es el origen y el nodo con ID mayor corresponde al destino. Internamente, el programa traduce estos valores y los hace válidos para ambos sentidos.
- Guardar el fichero con extensión ".txt"
- En este caso, sólo se trabaja con tres tipologías de poblaciones de paso. Tipo 1, Tipo 2 y Tipo 3. Cualquier valor fuera de este rango puede provocar resultados inesperados o fallos de programa.

3. Fichero de zonas naturales

El tratamiento que hace el programa del fichero de zonas naturales es muy similar al de las poblaciones de paso. Existen pequeñas diferencias, ya que en lugar del número de habitantes ahora se trata de los Kilómetros de la zona natural que son atravesados por la carretera. De este modo se logra obtener un coste total en función de la distancia atravesada, lo cual permite un ajuste mejor de los costes finales correspondientes a cada arista afectada.

La manera de trabajar del DSS con las zonas naturales puede resumirse así:

Originalmente tenemos una ruta sencilla, la más corta, que parte del almacén y recorre A, B y C y regresa finalmente al almacén.

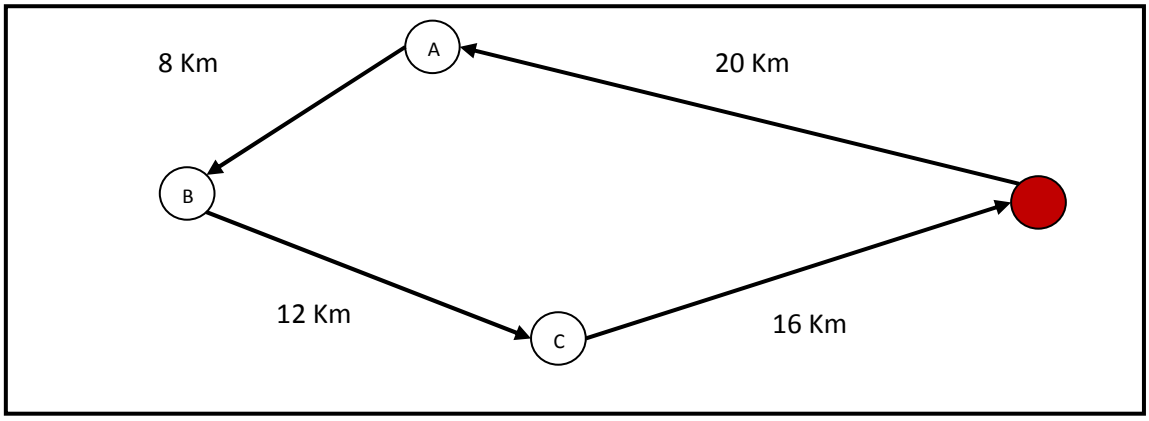


Ilustración 4.17. Elaboración propia.

El coste Kilométrico de esta primera solución es simplemente la suma de las distancias de las aristas de la ruta:

Coste_Kilométrico_A_B_C = 20 + 8 + 12 + 16 = 56 Km

El problema que queremos explicar se produce cuando en alguna de las aristas de la ruta original se interpone una zona de interés ecológico. El caso puede ilustrarse de la siguiente forma:

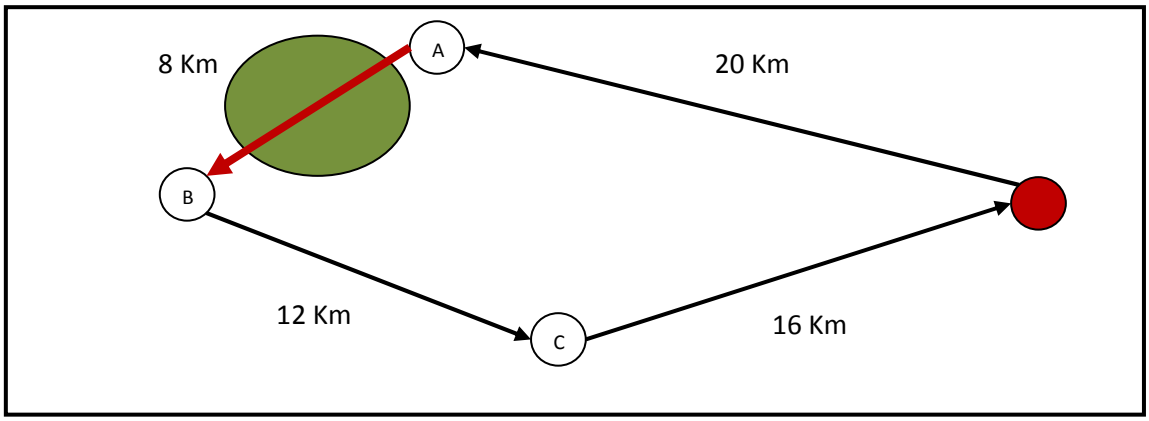


Ilustración 4.18. Elaboración propia.

Es en este momento cuando el DSS tiene que recalcular el coste de la ruta, concretamente el coste de la Arista A-B. El coste total de esta arista se calcula sumando el coste kilométrico inicial más el coste que supone atravesar x Kilómetros de zona natural del tipo y.

Coste_total_A_B = Coste-A-B + (kilómetros* Ajuste_Tipo * Ajuste_de_corrección)

Con lo que el coste de la ruta A_B_C quedaría:

Coste_total_A_B_C = Coste_total_A_B + 12 + 16

Una vez recalculado el coste de todas las aristas, añadiendo los costes adicionales por zonas naturales donde corresponda, el algoritmo vuelve a buscar cual es la ruta más corta. El caso que cabría esperar, siempre en función de los ajustes asignados, es algo parecido al siguiente grafo:

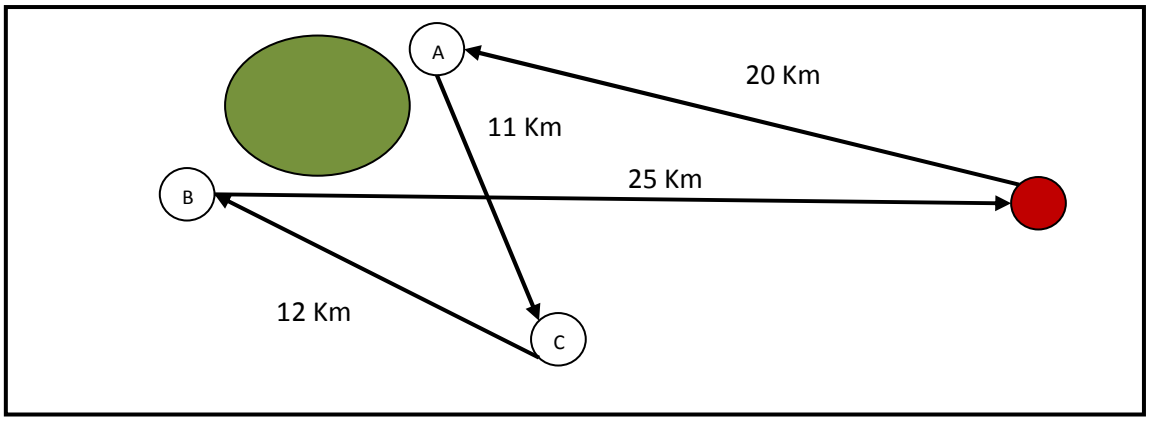


Ilustración 4.19. Elaboración propia.

Esta última ruta será seleccionada y mostrada como la solución del problema, siempre y cuando su coste total (añadiendo el coste de atravesar la zona natural) sea menor al coste total de la ruta A_B_C. El coste de esta nueva ruta será:

Coste_total_A_C_B = 20 + 11 + 12 + 25 = 68 Km

En cuanto a los valores de ajuste, el ajuste de corrección no es más que un valor que el usuario puede cambiar fácilmente en tiempo de ejecución desde la pantalla principal de la interfaz del programa:

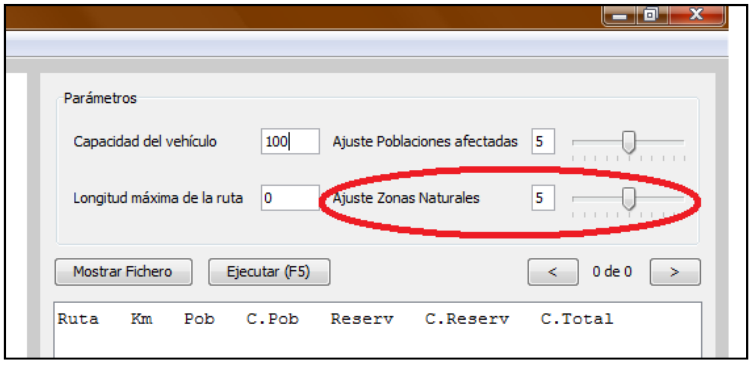


Ilustración 4.11. Valores de ajuste. GR-DSS

De esta manera, podemos ir cambiando la relevancia que le damos a los distintos ficheros cargados en el mapa. Dando el valor 0, el programa trabajará como si no existieran zonas naturales en el problema, y por tanto, veremos cómo las rutas generadas probablemente pasen por las aéreas correspondientes a dichas zonas naturales.

A medida que aumentamos el valor del Ajuste de zonas naturales, las nuevas rutas tendrán cada vez mayor facilidad para evitar estas zonas, buscando alternativas distintas que satisfagan estos nuevos requisitos.

Al igual que ocurre con los pueblos de paso, esto da al usuario la opción de obtener una gran cantidad de soluciones diferentes con un tiempo mínimo, ya que no es necesario cargar ningún fichero de nuevo, basta con pulsar el botón de Ejecutar y se generará una nueva solución con los valores de ajuste seleccionados.

En cuanto al Ajuste del Tipo de Zona natural, me he basado en las calificaciones que se hacen a nivel nacional. En España se hace una distinción entre parque natural y parque nacional. Al contrario de lo que suele pensarse, las categorías de espacios naturales protegidos en España (Ley 4/1989) no se basan en niveles mayores o menores de protección, por lo que el Parque Nacional "no" es la figura de mayor protección. Se basa en sus funciones y características que son:

- Parques: áreas naturales, poco transformadas por la explotación u ocupación humana que, en razón a la belleza de sus paisajes, la representatividad de sus ecosistemas o de su flora, de su fauna o de sus formaciones geomorfológicas, poseen unos valores ecológicos, estéticos, educativos y científicos cuya conservación merece atención preferente. Un Parque Nacional lo es por ser de interés nacional en razón de que sea representativo del patrimonio natural y de que incluya alguno de los principales sistemas naturales españoles.
- Reservas Naturales: espacios naturales cuya creación tiene como finalidad la protección de ecosistemas, comunidades o elementos biológicos que, por su rareza, fragilidad, importancia o singularidad merecen una valoración especial.
- Monumentos Naturales: espacios o elementos de la naturaleza constituidos básicamente por formaciones de notoria singularidad, rareza o belleza, que merecen ser objeto de una protección especial.
- Paisajes Protegidos: lugares concretos del medio natural que, por sus valores estéticos y culturales, sean merecedores de una protección especial.

Sin embargo, al igual que en el fichero de poblaciones, se ha establecido una tipología de zona natural ajustada a las necesidades del ejemplo de Navarra, y en este caso sólo se han establecido dos valores posibles:

- **Tipo 1:** En este tipo se incluyen parques naturales, o zonas de especial interés ecológico. Son zonas protegidas por la legislación, pero donde algunas actividades humanas están permitidas, al menos en zonas restringidas. Para el ejemplo de Navarra se han incluido en esta categoría El parque natural de la sierra de Urbasa y Andía y el Señorío de Bértiz.
- **Tipo 2:** Este tipo representa zonas que por su fragilidad o por su peculiaridad, ya sea en flora o en fauna, pueden clasificarse en un escalón superior de relevancia. Su clasificación y reconocimiento se hace a nivel mundial por los organismos pertinentes, como puede ser la UNESCO. En este tipo se ha incluido Las Bardenas Reales, ya que es un parque natural donde existen varias zonas concretas declaradas Reserva de la Biosfera desde el año 2000.

El fichero con la información de las tres zonas naturales seleccionadas para el ejemplo de Navarra contiene los siguientes datos:

ID Origen	ID Destino	Nombre Origen	Nombre Destino	Km	Tipo	Coordenada X	Coordenada Y
16	17	Alsasua	Estella	12	1	-40	1
23	24	Carcastillo	Tudela	10	2	10	-65
2	10	Elizondo	Sunbilla	7	1	3	39
6	9	Irurita	Sunbilla	7	1	3	39

Tabla 4. Fichero de Zonas naturales.

En este caso, las tres zonas naturales escogidas afectan a cuatro aristas de las posibles rutas. Al igual que con el fichero de poblaciones, estas zonas naturales han sido escogidas con el objetivo de hacer el ejemplo ilustrativo, por lo que otras zonas de relevancia ecológica de la geografía Navarra se han obviado y no han sido incluidas en el supuesto.

Del mismo modo que ocurre con los anteriores ficheros, el DSS es capaz de trabajar con el número de zonas naturales cualquiera, repercutiendo únicamente en los recursos necesarios para el cómputo de las rutas.

Las cuatro aristas que se ven afectadas por estas tres zonas naturales son:

a. *Alsasua—Estella (Urbasa y Andía)*
b. *Carcastillo—Tudela (Bardenas Reales)*
c. *Elizondo—Sunbilla (Señorío de Bértiz)*
d. *Irurita—Sunbilla (Señorío de Bértiz)*

Las zonas naturales anteriormente mencionadas, están distribuidas geográficamente de la siguiente manera:

Ilustración 4.12. Distribución de las zonas naturales en el mapa.

La razón principal de no seleccionar más zonas de interés ecológico es la de simplificar el ejemplo. Si bien existen otros parques naturales en la geografía Navarra, éstos no están cerca de los nodos clientes o su ubicación complica el resultado de las rutas, perjudicando el objetivo de mostrar los resultados de forma sencilla y rápida.

Su representación en el visor gráfico del DSS queda del siguiente modo:

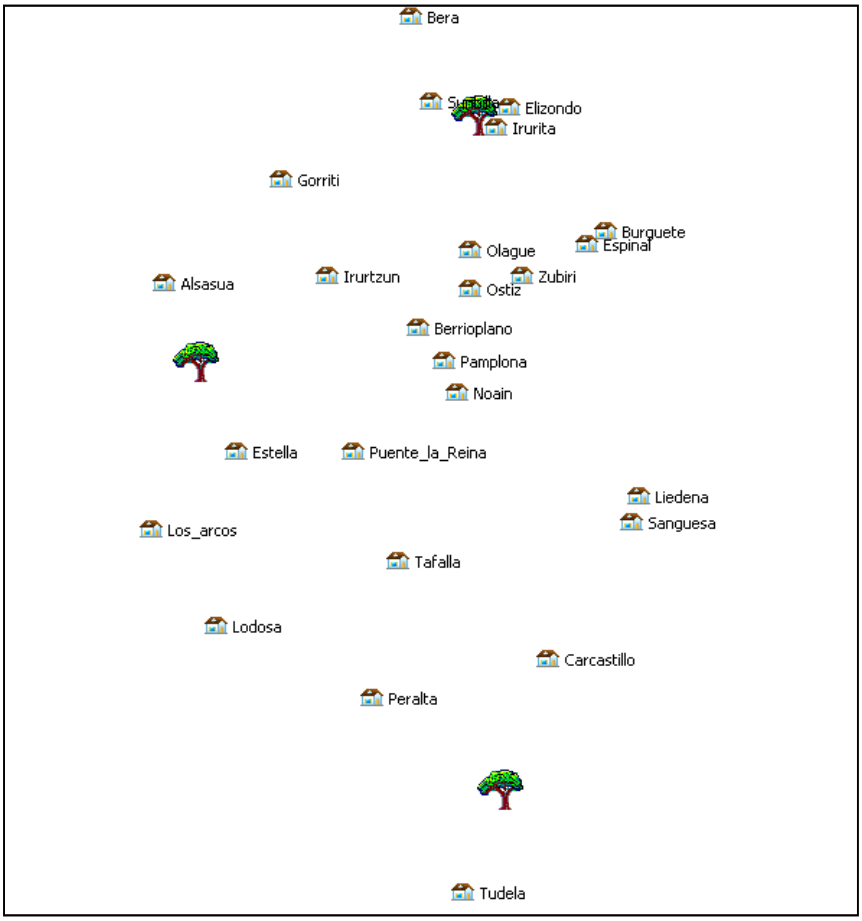


Ilustración 4.13. Distribución de las zonas naturales. Visor del GR-DSS

En este caso, las zonas naturales son representadas con la imagen de un árbol. Para la representación gráfica no se ha tenido en cuenta la extensión real de cada zona natural, simplemente se ha ubicado la imagen en un punto que correspondería a la ubicación real de la zona.

A efectos de cálculo, el DSS sólo tiene en cuenta los kilómetros y el tipo de zona natural, no siendo necesario más información para diseñar las rutas.

Requisitos del fichero de zonas naturales:

- Los nombres de las localidades han de formar una única cadena de caracteres. Es decir, los nombres de localidades formados por más de una palabra han de seguir el siguiente formato: Puente_la_Reina.
- Entre columnas de valores pueden introducirse tantos espacios en blanco o tabuladores como se quiera.
- Hay que respetar el orden de las columnas, en caso de introducir los valores desordenados, el programa falseará los resultados o simplemente fallará al intentar cargar el fichero.
- Tener cuidado en no dejar líneas intermedias vacías ni tampoco al final del fichero.
- El orden del ID_Origen y el del ID_Destino han de ser de menor a mayor. Con los dos extremos se describe la arista que los une. Por motivos de programación, es necesario introducir de este modo las aristas, el nodo con ID menor es el origen y el nodo con ID mayor corresponde al destino. Internamente, el programa traduce estos valores y los hace válidos para ambos sentidos.
- Guardar el fichero con extensión ".txt"
- En este caso, sólo se trabaja con dos tipologías de zonas naturales. Tipo 1 y Tipo 2.

4.3.2. Interfaz gráfica

En este apartado se explica el diseño de la interfaz gráfica que se ha elegido para el DSS. En este caso se han usado las herramientas gráficas de diseño que proporciona el entorno de programación de NetBeans.

En este caso, existen dos paneles principales, en los cuales aparecerá representada la mayoría de la información, tanto entradas como salidas.

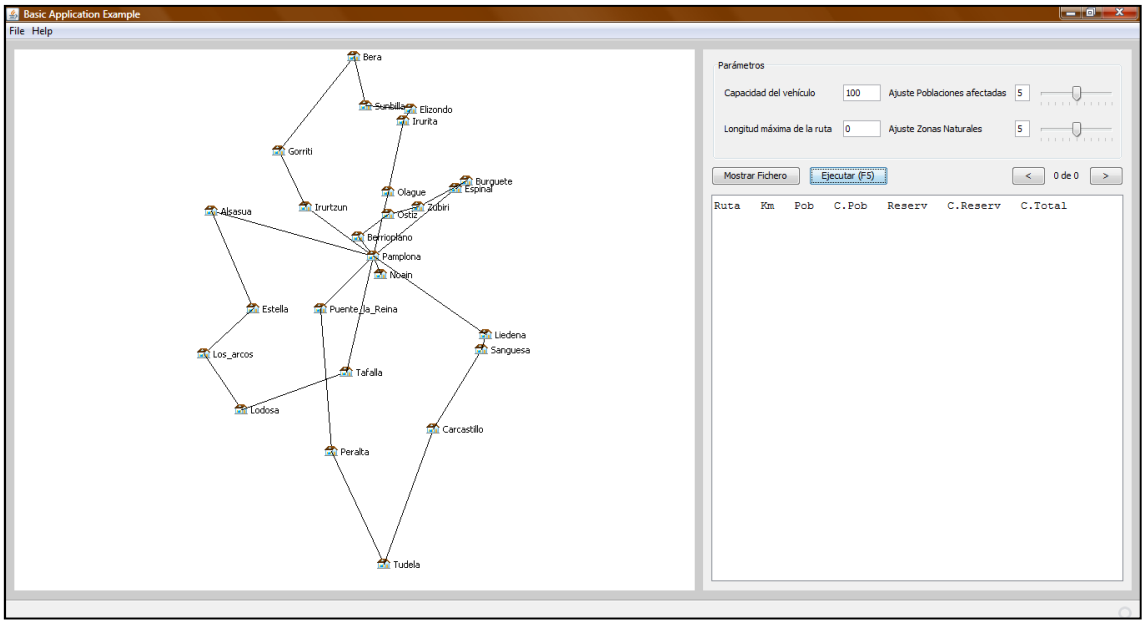


Ilustración 4.20. Interfaz principal. GR-DSS

En el caso del panel de la izquierda, el más grande de los dos, aparecerán representados los nodos cliente del fichero de nodos seleccionado.

Para el correcto escalado y centrado de la imagen resultante, se ha tenido que diseñar una serie de métodos que, a través de operaciones matemáticas, permiten recalcular la escala necesaria cada vez que se redimensiona la ventana.

En el proceso de mostrar la interfaz, así como de dibujar los diferentes componentes intervienen varias clases. Entre ellas cabe destacar la clase DSSView o la JPaintPanel.

En concreto, el método que obtiene los valores actuales de anchura y altura del panel es el siguiente y pertenece a la clase JPaintPanel:

```
private float[] getEscalaYDesplazamiento() {

        int MaxX = Integer.MIN_VALUE;
        int MinX = Integer.MAX_VALUE;
        int MaxY = Integer.MIN_VALUE;
        int MinY = Integer.MAX_VALUE;
        int AnchuraPanel = this.getWidth();
        int AlturaPanel = this.getHeight() - Margen;

        //Hallamos el mayor y menor valor de X e Y
        for (Node d : Solucion.getNodes()) {
            if (d.getX() > MaxX) {
                MaxX = d.getX();
            }
            if (d.getY() > MaxY) {
                MaxY = d.getY();
            }
            if (d.getX() < MinX) {
                MinX = d.getX();
            }
            if (d.getY() < MinY) {
                MinY = d.getY();
            }
        }
        //Calculamos el desplazamiento para encuadrar el dibujo en el origen
        //Tomaremos el negativo del minimo de manera que si es negativo le
        //sumaremos su valor para que quede en el 0 y si es positivo se lo
        //restaremos
        int DesplazamientoX = -MinX;
        int DesplazamientoY = -MinY;
        //Calculamos la escala para aprovechar todo el espacio
        //Tomamos el menor de las escalas en X e Y utilizandola en las dos
        //coordenadas la misma para no deformar la imagen
        float Escala = Math.min((float)AnchuraPanel / (MaxX + DesplazamientoX), (float)AlturaPanel / (MaxY + DesplazamientoY));

        float[] param = new float[5];
        param[0] = Escala;
        param[1] = DesplazamientoX;
        param[2] = DesplazamientoY;
        param[3] = MaxX;
        param[4] = MaxY;
        return param;
    }
```

Código 4.3. Método para centrar y escalar las imágenes. GR-DSS

A su vez, cada componente que vaya a ser dibujado, conocerá estos parámetros para obtener su correcta ubicación. Esto es gracias a que las clases base de los nodos, de los pueblos de paso y de las zonas naturales implementan la interfaz "*Drawable*", que contiene un único método *Draw,* el cual contiene los parámetros (alturaPanel, AnchuraPanel, Escala, DesplazaminetoX, DesplazamientoY, MaxXescalada y MaxYescalada) para la correcta ubicación del objeto en cuestión.

```
package Interfaces;

import java.awt.Graphics;

/**...*/
public interface DrawAble {

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
            int MaxXEscalada,int MaxYEscalada);

}
```

Código 4.4. Interfaz DraWable. GR-DSS

Por otro lado, en el panel de la zona inferior izquierda se muestra la información de salida obtenida una vez es ejecutado el DSS.

La información mostrada en este panel al inicio es el nombre de las siguientes columnas:

Ruta ID: El programa asigna internamente un identificador numérico a las diferentes rutas que son creadas durante su ejecución. Una vez finalizado, el programa muestra tan sólo las mejores rutas.

Una vez obtenida una ruta solución, ésta se compone de varias subrutas, cada una de las cuales parte del almacén, recorre una serie de nodos y vuelve al almacén. Éstas subrutas también se identifican con un ID, y se muestran los diferentes costes asociados a cada una de ellas.

Km: los kilómetros recorridos por cada subruta.

Pob: Mostrará el número de habitantes de las poblaciones atravesadas por cada ruta.

C. Pob: Es el coste correspondiente a cada pueblo de paso que es atravesado por la ruta. Dependerá del número de habitantes, del tipo del pueblo y del parámetro de ajuste seleccionado.

Reserv: Muestra los kilómetros de reserva atravesados por cada subruta.

C. Reserv: Es el coste asociado a cada zona natural atravesada por una ruta. Dependerá del número de Kilómetros atravesados, así como al tipo de reserva y al valor del parámetro de ajuste seleccionado.

C. Total: La suma de los costes asociados a cada subruta. Coste kilométrico + Coste de pueblos de paso + Coste de zonas naturales.

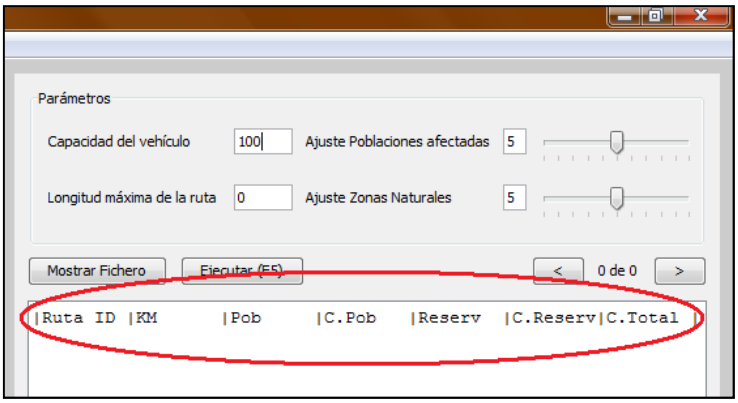


Ilustración 4.21. Salida de información. GR-DSS

La forma en la que se muestra la información de salida será analizada en profundidad más adelante en el apartado de salidas.

4.3.3. Ejecución del programa

Como se explicó en el capítulo 1, en la parte referente al SimuRoute, el algoritmo se basa en las coordenadas de los nodos clientes para crear una matriz de costes.

Esta matriz contiene las distancias (en línea recta) entre cada par de nodos. De esta forma, el algoritmo obtiene los costes de cada subruta calculada.

Para el DSS de este proyecto ha habido que realizar algunas modificaciones en el algoritmo SimuRoute, y entre ellas, la construcción de dos matrices de costes auxiliares.

Cada una de las dos matrices auxiliares corresponde a los pueblos de paso o a las zonas naturales y contienen los costes que suponen los pueblos de paso o las zonas naturales al ser atravesados, respectivamente.

El modo en que se calculan estos valores es sencillo. La estructura del fichero de entrada obliga a que cada zona natural o pueblo de paso corresponda a una arista existente entre dos nodos cliente, A y B.

De este modo, la nueva matriz auxiliar asocia a la posición A, B (y también la posición B, A) el coste obtenido de combinar matemáticamente los valores de coste que correspondan en cada caso. Para la matriz de costes de pueblos de paso, será resultado de operar con el número de habitantes y tipo de pueblo. En el caso de la matriz de costes de zonas naturales, lo hace con los kilómetros atravesados y el tipo.

Una vez creadas las dos matrices auxiliares, los valores de éstas se suman a los valores iniciales de la matriz de costes original.

La forma en la que se hace esta suma de valores a la matriz de costes está en función de los parámetros de ejecución que el usuario ha seleccionado.

Este proceso se puede explicar mediante un ejemplo:

Suponemos que el usuario da un valor de ajuste 0 al parámetro de ajuste de pueblos de paso (en la interfaz principal). El algoritmo calculará la matriz de costes inicial y calculará la matriz de costes asociada a los pueblos de paso.

Cuando llega el momento de sumar los valores, los costes de los pueblos de paso son multiplicados por 0 antes de ser sumados. Obviamente, la matriz de costes final resultante es igual a la original, ya que no hemos sumado nada nuevo.

Por el contrario, si seleccionamos otro valor para el ajuste de pueblos de paso, dentro de la escala permitida (del 0 al 10) en la interfaz, los valores de la matriz de costes de los pueblos de paso son multiplicados por dicho parámetro antes de sumarse a los de la matriz de costes original.

Siempre que el valor sea mayor que 0, la matriz de costes final será distinta a la original, teniendo unos costes mayores entre aquéllos nodos entre los que existe un pueblo de paso o una zona natural.

El proceso general de creación de las matrices se puede apreciar en el siguiente diagrama:

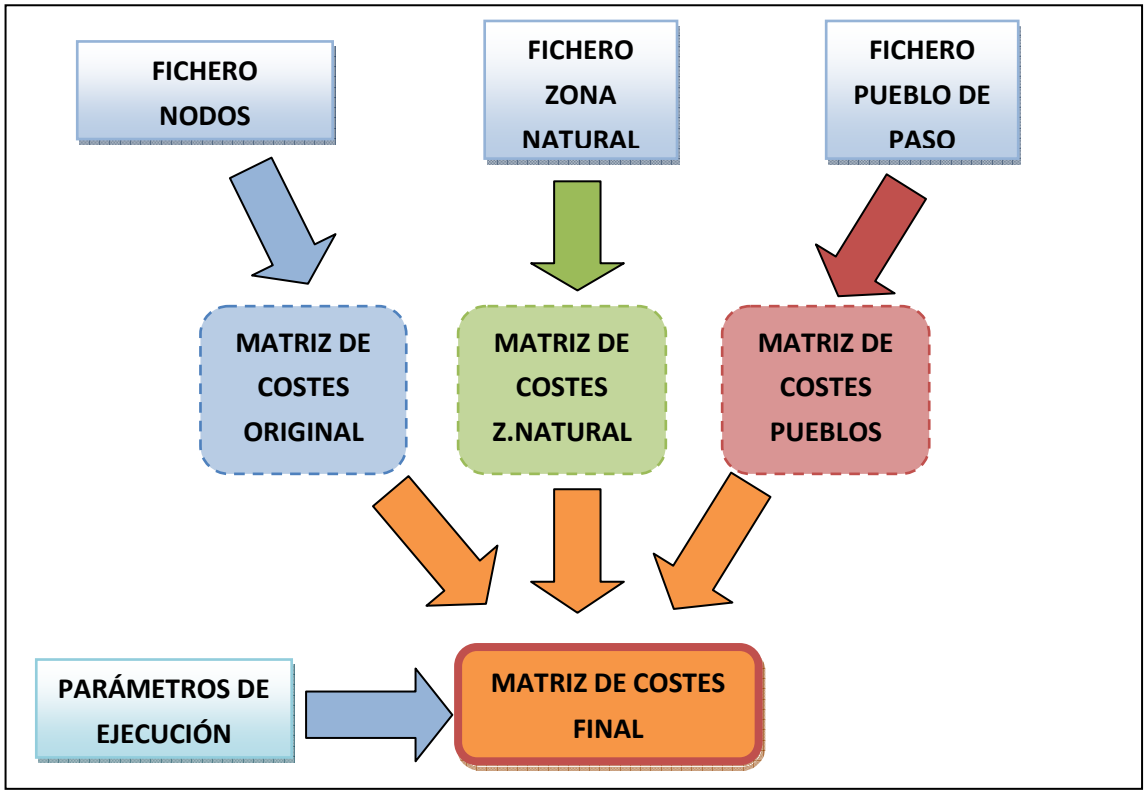


Gráfico 4.1. Creación de las matrices de costes. GR-DSS

Como se aprecia en el diagrama, las matrices de costes asociados a los pueblos de paso y a las zonas naturales se crean a partir de la información obtenida de los ficheros de entrada correspondientes.

La matriz de costes original se crea a partir de las coordenadas de los nodos descritos en el fichero de entrada de nodos cliente.

Los parámetros de ejecución seleccionados por el usuario influyen a la hora de crear la matriz de costes final.

Ahora se hace correr al algoritmo pero con la nueva matriz de costes final, de modo que calcule las rutas teniendo en cuenta los costes añadidos al atravesar zonas naturales o pueblos de paso.

El código encargado de crear las matrices de costes asociadas a los pueblos de paso y a las zonas naturales es el siguiente:

```
    public void setPoblacionesMatrix(ArrayList<BasePoblacionDePaso> poblaciones) {
        for (BasePoblacionDePaso poblacion : poblaciones) {
            int IdOrigen = poblacion.getIdOrigen();
            int IdDestino = poblacion.getIdDestino();
            int tipo = poblacion.getTipo() - 1;
            int pob = poblacion.getPoblacion();
            this.poblacionesMatrix[IdOrigen][IdDestino] = (long) (pob * Parametros.ModificadorPoblacion[tipo] * Parametros.
            this.poblacionesMatrix[IdDestino][IdOrigen] = this.poblacionesMatrix[IdOrigen][IdDestino];

            this.HabPoblacion[IdOrigen][IdDestino] = pob;
            this.HabPoblacion[IdDestino][IdOrigen] = pob;
        }
    }

    public long[][] getZonasMatrix() {...}

    public void setZonasMatrix(ArrayList<BaseZonaNatural> zonas) {
        for (BaseZonaNatural zona : zonas) {
            int IdOrigen = zona.getIdOrigen();
            int IdDestino = zona.getIdDestino();
            int KM = zona.getKmZonaNatural();
            int tipo = zona.getTipoZonaNatural() - 1;
            this.zonasMatrix[IdOrigen][IdDestino] = (long) (KM * Parametros.ModificadorZona[tipo] * Parametros.intFactor);
            this.zonasMatrix[IdDestino][IdOrigen] = this.zonasMatrix[IdOrigen][IdDestino];

            this.KMReserva[IdOrigen][IdDestino] = KM;
            this.KMReserva[IdDestino][IdOrigen] = KM;
        }
    }
```

Código 4.5. Creación de las matrices de costes (pueblos de paso y zonas naturales)

Estos dos métodos leen los ArrayList correspondientes a los pueblos y a las zonas naturales y van completando las matrices correspondientes (*poblacionesMatrix* y *zonasMatrix*)

Para ello se multiplican los kilómetros por el tipo, en el caso de las zonas naturales y la población por el tipo para la matriz de costes de los pueblos de paso.

El código encargado de crear la matriz de costes final es el siguiente:

```
//Construye la matriz de costes teniendo en cuenta el ajuste
//Introducido en el menu de parámetros. (jSlider)
public long[][] getCostMatrix() {
    long[][] costMatrix = new long[dim][dim];
    for (int i = 0; i < dim; i++) {
        for (int j = 0; j < dim; j++) {
            costMatrix[i][j] = this.distanciasMatrix[i][j] + Math.round(Parametros.AjusteReservas * this.zonasMatrix[i][j])
                    + Math.round(Parametros.AjustePoblaciones * this.poblacionesMatrix[i][j]);
        }
    }
    return costMatrix;
}
```

Código 4.6. Matriz de costes final

Se puede ver cómo es aquí donde el ajuste introducido por el usuario multiplica a los valores correspondientes a cada matriz auxiliar.

Finalmente, se devuelve la matriz costMatrix con la que el algoritmo SimuRoute es capaz de empezar a calcular las mejores rutas.

4.3.4. Salidas

La salida de información se da de dos formas:

Por un lado, la información mostrada en pantalla, en el propio visor de la interfaz del DSS. En este caso encontramos la información de las soluciones en dos zonas diferenciadas. La primera consiste en información gráfica y se da en el panel principal de la interfaz. En él, las rutas solución obtenidas tras la ejecución son dibujadas, de manera que se pintan las aristas que unen los nodos clientes según el orden encontrado por el algoritmo. La otra zona donde se muestran las soluciones es el panel inferior derecho. En este panel se resumen los datos de cada ruta solución.

Por otro lado, la información detallada de cada solución es volcada en un fichero de texto. De este modo se consigue guardar las soluciones obtenidas en cada ejecución de una forma permanente.

4.3.4.1. Por pantalla

Como se ha dicho, el programa cuenta con dos zonas donde mostrar la información obtenida tras la ejecución. Ambas zonas se ven a la vez, ya que forman parte de la interfaz principal.

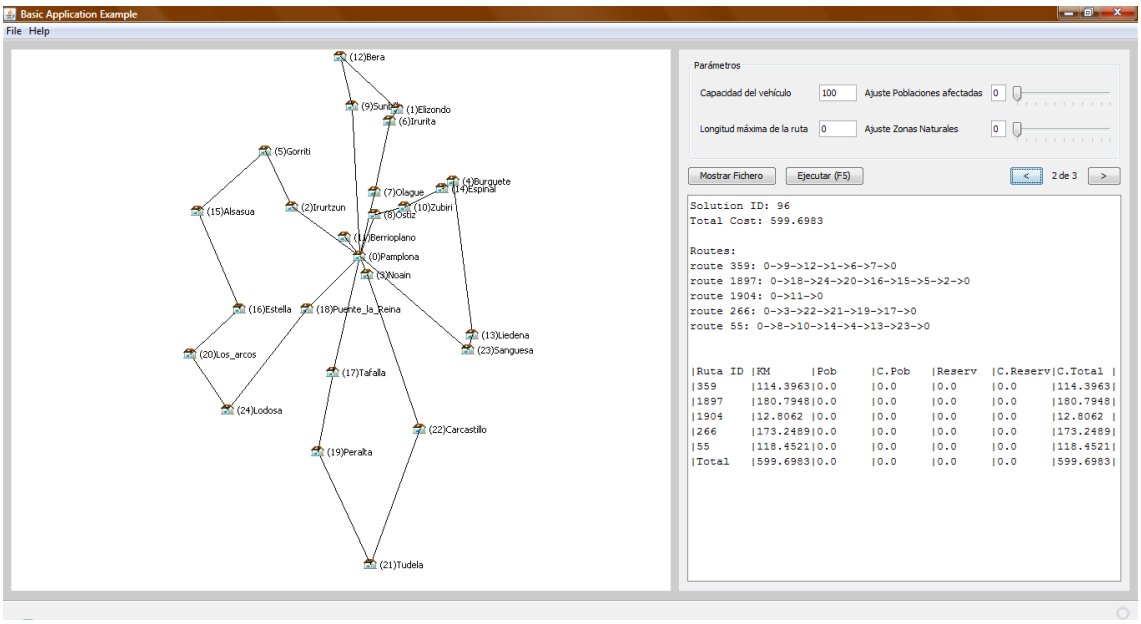


Ilustración 4.22. Vista de las soluciones en la interfaz principal. GR-DSS

Se ha diseñado la interfaz de esta manera con el objetivo de que el usuario sea capaz de recibir de una forma clara y rápida toda la información relevante de las distintas soluciones.

Ambas zonas están interrelacionadas, ya que muestran la información referente a una misma solución, y si la solución cambia, la información de las dos zonas cambia al mismo tiempo.

El panel principal muestra de manera gráfica las soluciones obtenidas. Lo hace dibujando las aristas correspondientes a cada solución uniendo los nodos clientes en el orden obtenido por el algoritmo.

Los nodos cliente son representados por un pequeño icono con forma de casa, y a su lado se muestra el nombre de la población así como el ID correspondiente a cada uno de ellos. Así podemos ver el orden que siguen las rutas, así como los nodos visitados por cada una de ellas.

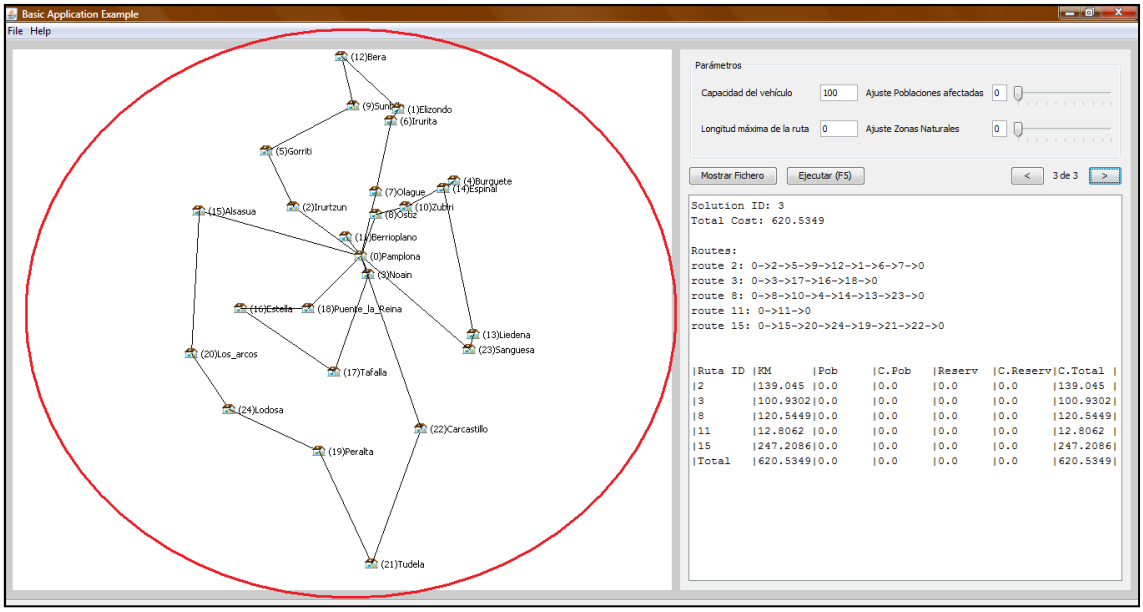


Ilustración 4.23. Panel principal. Vista de las rutas. GR-DSS

La interfaz incluye dos botones que permiten navegar entre las soluciones obtenidas por el algoritmo. Esta es una de las ventajas de este algoritmo utilizado, que permite obtener en una sola ejecución varias de las mejores soluciones obtenidas.

Esto nos permite comparar las soluciones obtenidas, viendo como cada vez que lo hacemos, la información que aparece en ambos visores cambia para hacer referencia a la solución seleccionada.

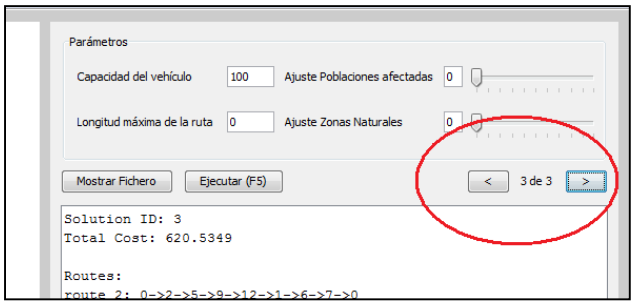


Ilustración 4.24. Botones de navegación entre soluciones. GR-DSS

En el panel secundario, en la zona inferior derecha, se muestra un resumen de la información general de cada solución.

Primero se muestra el ID correspondiente a cada solución, de este modo podemos identificar cada una de las soluciones, lo que facilita su comparación posterior.

A continuación se muestra el coste total de la solución. Este coste corresponde al coste de la ruta sin tener en cuenta los pueblos de paso atravesados ni las zonas naturales. Por lo tanto, cuando los parámetros de ajuste de ambos factores sean 0, los costes inicial y final coincidirán.

Más abajo se describen las diferentes subrutas que forman la solución. Se hace indicando el Id de los nodos clientes que sigue cada una de ellas.

Después se muestra una tabla que contiene la siguiente información para cada subruta:

Km: los kilómetros recorridos por cada subruta.

Pob: Mostrará el número de habitantes de las poblaciones atravesadas por cada ruta.

C. Pob: Es el coste correspondiente a cada pueblo de paso que es atravesado por la ruta. Dependerá del número de habitantes, del tipo del pueblo y del parámetro de ajuste seleccionado.

Reserv: Muestra los kilómetros de reserva atravesados por cada subruta.

C. Reserv: Es el coste asociado a cada zona natural atravesada por una ruta. Dependerá del número de Kilómetros atravesados, así como al tipo de reserva y al valor del parámetro de ajuste seleccionado.

Y finalmente, se muestra en la última fila los valores totales de cada uno de los anteriores parámetros así como el coste total:

C. Total: La suma de los costes asociados a cada subruta. Coste kilométrico + Coste de pueblos de paso + Coste de zonas naturales.

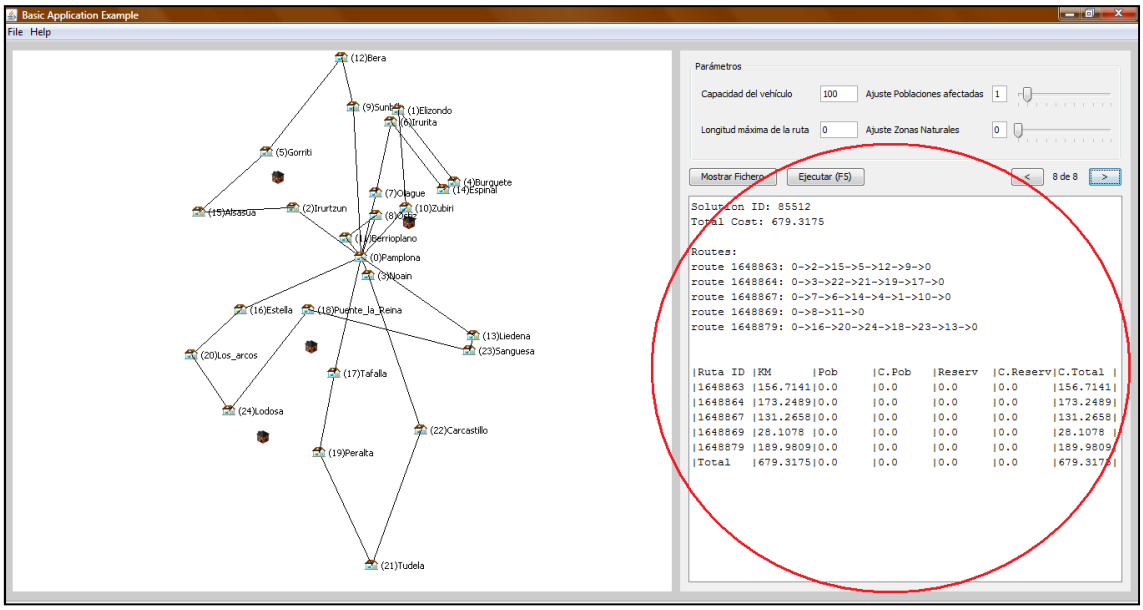


Ilustración 4.25. Vista detalle de soluciones. Panel secundario. GR-DSS

La combinación de los dos paneles de la interfaz, da al usuario toda la información correspondiente a cada solución de una manera clara y eficaz. Podemos ver de un solo vistazo las soluciones obtenidas, así como los costes asociados a cada una de ellas.

Esto hace posible la corrección de criterios en el momento. Por ejemplo, si localizamos algún factor de las soluciones que no interesa, como puede ser que las rutas solución atraviesan determinado pueblo X, basta con cambiar los ajustes seleccionados y repetir la ejecución.

De este modo, el usuario tiene la posibilidad de ir jugando con los distintos valores de ajuste hasta encontrar las soluciones más adecuadas a sus preferencias.

4.3.4.2. En fichero

La información devuelta por cada solución lograda, se guarda en un fichero de texto, para su posterior utilización.

De esta forma se consigue un método que permite mantener las soluciones de una forma permanente. En el futuro, pueden emplearse estos ficheros para realizar comparaciones y estudios, o simplemente, guardarse como justificantes de las rutas realizadas.

La información que se vuelca al fichero de salida es la prácticamente la misma que se muestra en el panel secundario, solo que se guardan juntas todas las soluciones resultado de una misma ejecución. Esto hace que tengamos un fichero de salida para cada ejecución realizada.

Por defecto, el fichero se guardará en la misma dirección en la que se encuentran los ficheros de entrada.

Una muestra de cómo queda un fichero de salida:

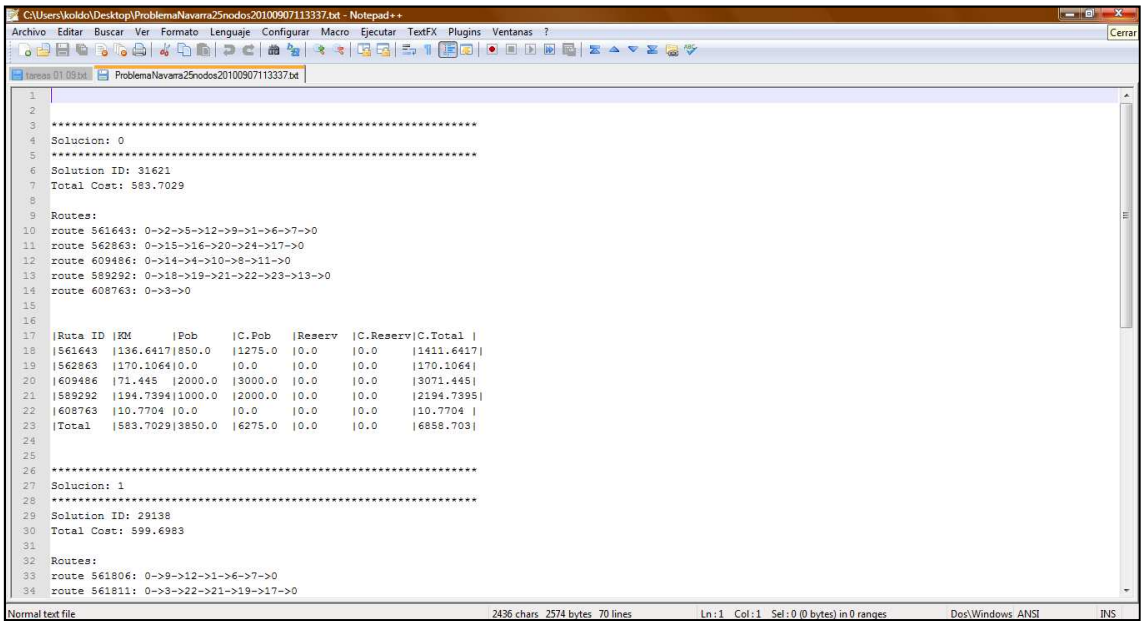


Ilustración 4.26. Vista fichero de soluciones. GR-DSS

El nombre de los ficheros de salida se corresponde con el nombre del fichero de entrada de los nodos cliente, añadiendo la fecha y la hora exacta de la ejecución. Así se logra crear tantos ficheros como ejecuciones realizadas, ya que nunca dos ficheros solución van a tener el mismo nombre.

Este método de trabajo mantiene la seguridad del sistema, primando el guardado de todas las soluciones.

Un posterior proceso de cribado, podría permitir extraer aquella información de los ficheros que cumpliese unos determinados requisitos. Esto se explica en capítulo 6, Mejoras futuras.

5. PRUEBAS Y RESULTADOS OBTENIDOS

En este capítulo se detallan una serie de pruebas que se han realizado utilizando el GR-DSS con el ejemplo del supuesto de navarra, anteriormente explicado. (Véase capítulo 4)

Con este capítulo se pretende mostrar los resultados que se pueden obtener con el programa, dado un problema concreto. Así se quiere demostrar la efectividad del GR-DSS a la hora de encontrar una serie de rutas alternativas para un problema del la familia de los VRP.

Como ya se ha explicado, el propósito del programa es encontrar una serie de rutas que eviten, en la medida de lo posible, atravesar un conjunto de localizaciones que por sus características son especialmente sensibles a los efectos nocivos del tráfico.

Para ello, se explica a continuación como se han diseñado un conjunto de pruebas para el mismo problema, variando los parámetros de ajuste y obteniendo por tanto soluciones distintas, con distintos valores y costes.

En cada caso, al tener diferentes valores de ajuste, el objetivo buscado también es distinto. En unos se buscará minimizar el coste ocasionado por los habitantes de los pueblos de paso, en otros casos se intentará minimizar el coste generado por cruzar zonas naturales. Finalmente, se buscará cómo minimizar el coste generado por ambos criterios.

5.1. Condiciones de las pruebas

Los parámetros con los que se han realizado las pruebas han sido los programados por defecto en el programa. Para garantizar la integridad y la validez de las pruebas, en todas las ejecuciones se ha respetado la misma configuración de los parámetros de ejecución, excepto los valores de ajuste correspondientes a las poblaciones de paso y a las zonas naturales.Por supuesto, los ficheros de entrada utilizados para todo el conjunto de pruebas han sido siempre los mismos.

Se adjunta a esta memoria un soporte informático que contiene las carpetas correspondientes a cada una de las pruebas realizadas. En ellas existe una copia de los ficheros de entrada necesarios para cada caso, así como los ficheros de salida con las soluciones generadas en cada ejecución.

El conjunto de pruebas que se expone a continuación, no es más que una selección de alternativas, realizada con el objetivo de mostrar las capacidades que el GR-DSS puede poner en práctica.

El usuario del programa, que también se encuentra en el soporte informático adjunto a esta memoria, puede preparar fácilmente sus propias pruebas con distintos parámetros de entrada, incluso, puede variar o añadir información a los ficheros de entrada (manteniendo siempre la estructura preestablecida).

Los parámetros por defecto del programa son:

Capacidad del vehículo = 100
Longitud máxima de la ruta = 0

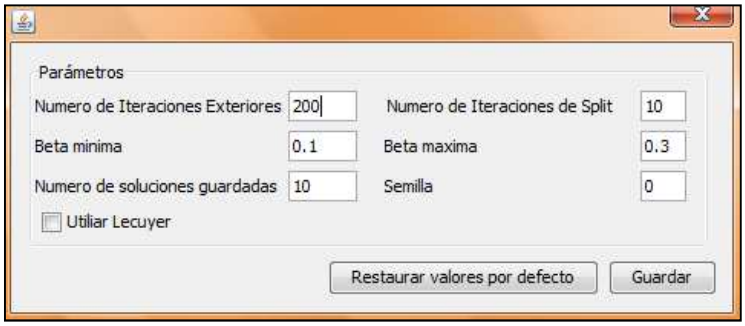


Imagen 5. 1

Número de iteraciones exteriores = 200
Número de iteraciones de Split = 10
Beta mínima = 0.1
Beta máxima = 0.3
Número de soluciones guardadas = 10
Semilla = 0

5.2. Pruebas y resultados

5.2.1. Prueba 1. Clientes

En esta prueba se tiene en cuenta solo el fichero de nodos clientes. Las soluciones, por tanto, son aquellas cuyo coste (en kilómetros) es menor.

En este caso, no tiene importancia el valor que demos a los ajustes de las poblaciones de paso y las zonas naturales. Esto es debido a que no se carga ningún fichero de pueblos ni de zonas y por tanto los costes asignados a sus respectivas matrices son todo ceros y no influyen en el resultado.

El GR-DSS muestra las mejores soluciones encontradas y las ordena empezando por la mejor o de menor coste total. A continuación, podemos ver el resto de soluciones con coste total creciente.

Para la prueba 1, el algoritmo ha seleccionado 3 soluciones:

Solución 1

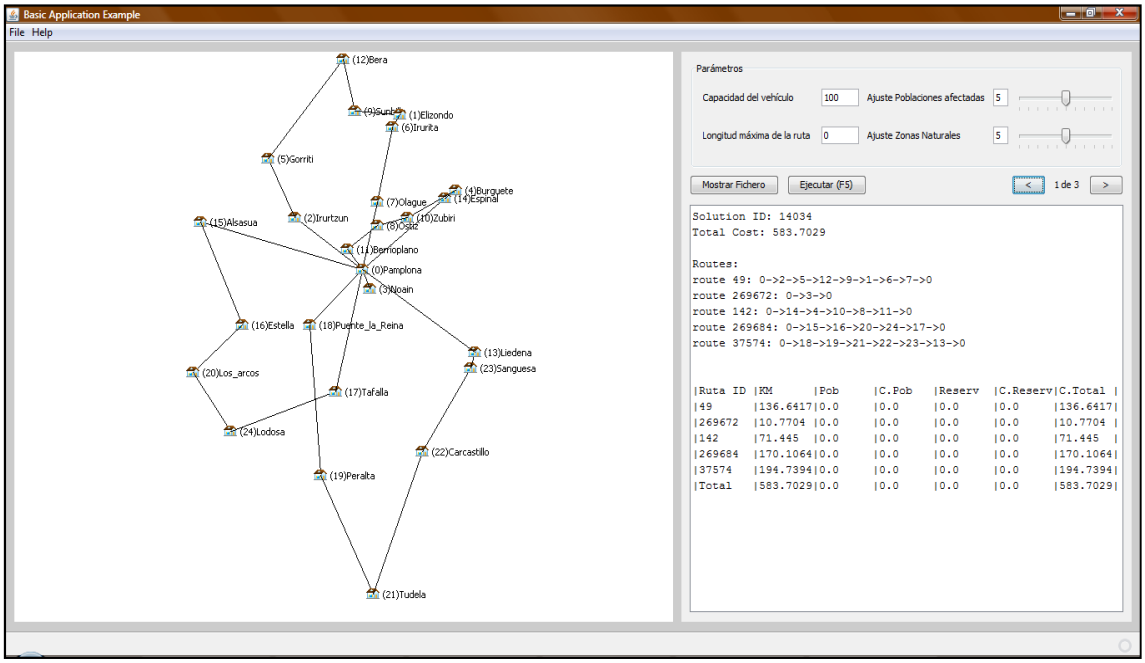


Imagen 5. 2

Esta es la mejor solución que el algoritmo SimuRoute original (sin ningún criterio ni restricción medioambiental) encontraría para el problema de Navarra.

En este caso, el coste total de la solución es el mismo que el coste kilométrico, 583.70

Solución 2

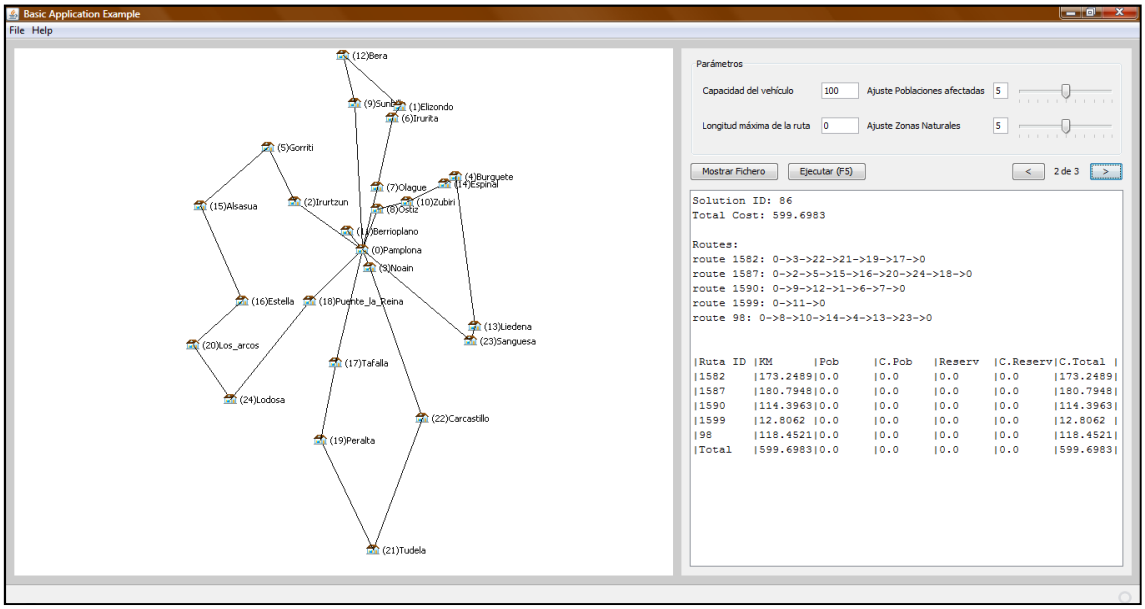


Imagen 5. 3

El coste de esta otra solución es 599.69

Solución 3

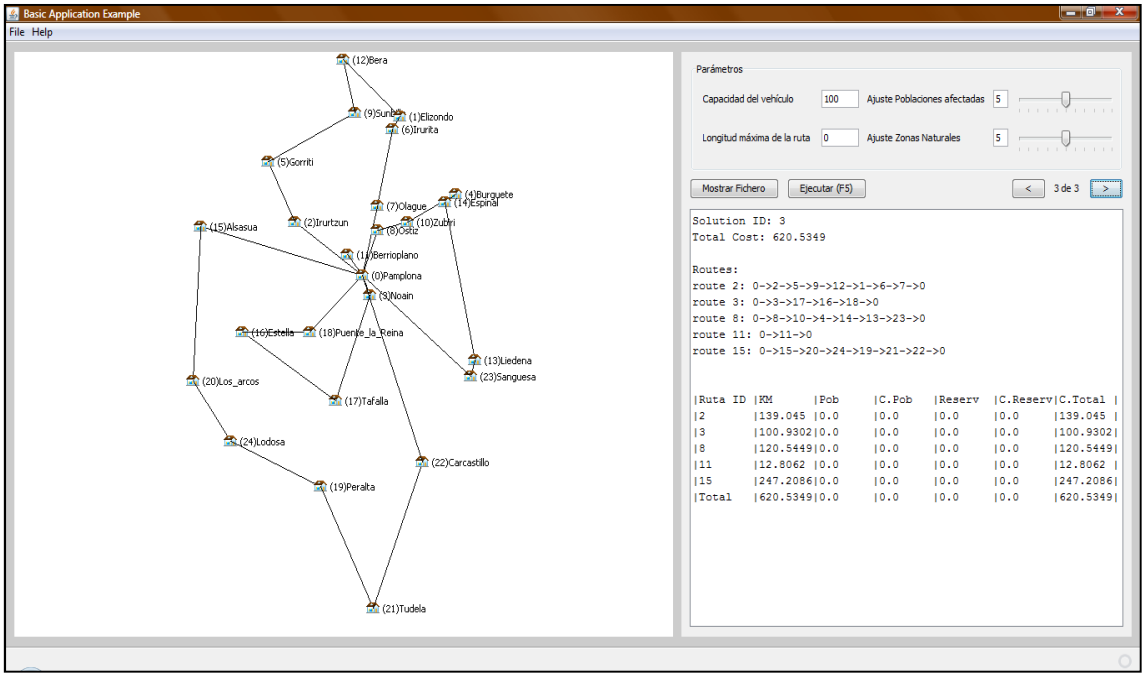


Imagen 5. 4

En este caso, el coste de la ruta solución es 620.53

5.2.2. Prueba 2. Clientes y Poblaciones de paso

En la siguiente prueba, se incluyen los pueblos de paso.

A continuación, la mejor solución con valor de ajuste para pueblos = 1.

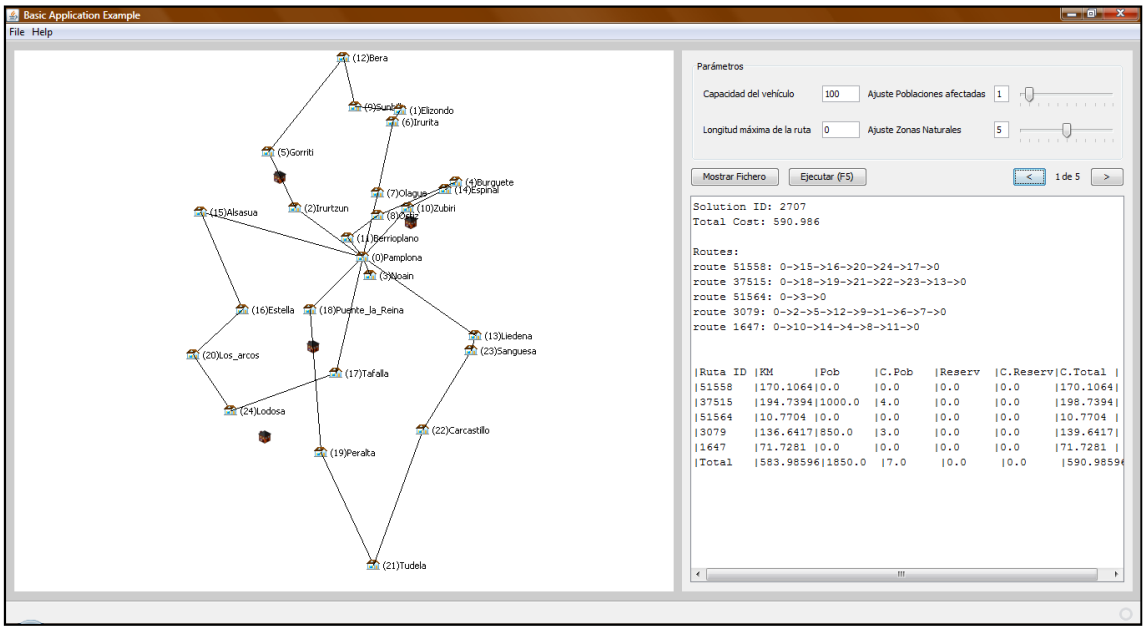


Imagen 5. 5

Coste total: 590.98

Coste kilométrico: 583.98

Coste Población: 7.0 (Larraga, Lekumberri)

Habitantes: 1850

Coste Reservas: 0

Kilómetros de reserva: 0

El programa nos devuelve 5 posibles soluciones.

Como ya se ha explicado, la mejor solución es la primera (ID 2707, en la imagen superior) y su coste total es de 590.986.

Comparado con la mejor solución original, 583.70, vemos que el coste se ha incrementado. En este caso, la ruta elegida sigue siendo la misma, solo que ahora se ha sumado el coste de atravesar 2 poblaciones (Larraga y Lekumberri)

Ahora probaremos cambiando el valor del ajuste de pueblos de paso a 5.

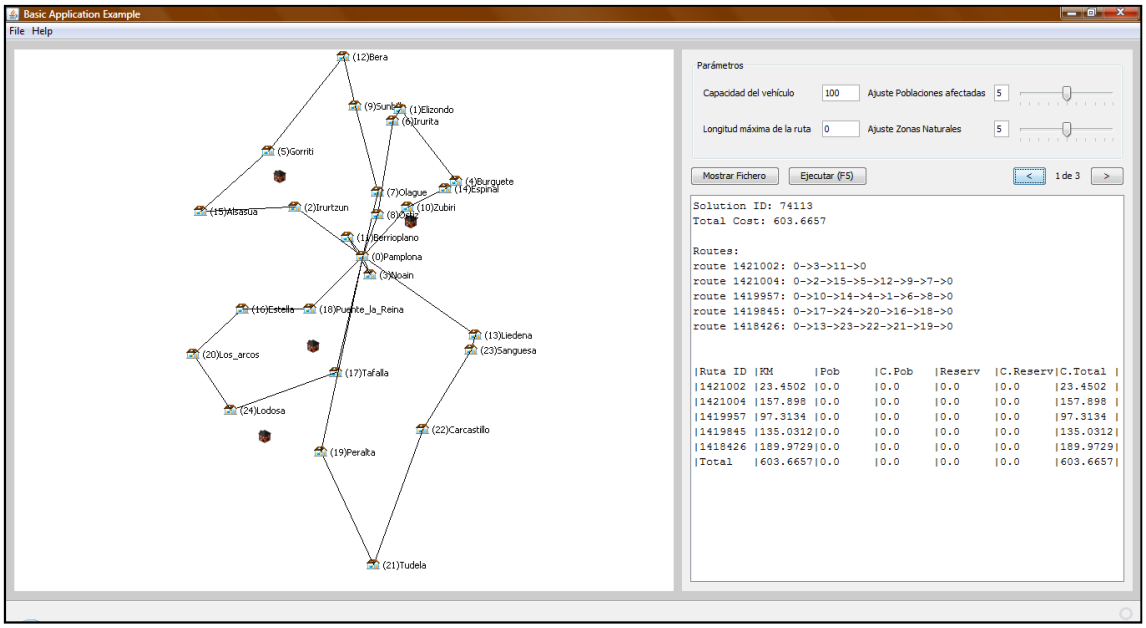


Imagen 5. 6

En este caso, el programa da tres posibles soluciones.

La mejor de ellas, tiene un coste total de 603.66, el cual se logra sin atravesar ninguna de las localidades de paso.

Coste total: 603.66

Coste kilométrico: 603.66

Coste Población: 0

Habitantes: 0

Coste Reservas: 0

Kilómetros de reserva: 0

Si volvemos a cambiar el valor del ajuste de pueblos de paso, en este caso a 10, el resultado conseguido se asemeja mucho al ejemplo anterior.

La mejor ruta obtenida es igual con el valor del ajuste igual a 5 que a 10.

Esto quiere decir que con un valor de 5 ya hemos conseguido encontrar rutas que evitan todos los pueblos de paso. No importará cuanto subamos el valor del ajuste, ya que las rutas generadas ya nunca pasan por los pueblos de paso seleccionados.

Un caso interesante se da al buscar un ajuste por el cual el algoritmo encuentra rutas en las cuales sale rentable atravesar alguno de los pueblos de paso.

Por ejemplo, con un valor de ajuste de pueblos de 3, el algoritmo encuentra una solución de este tipo. En concreto es la segunda solución, mostrada a continuación:

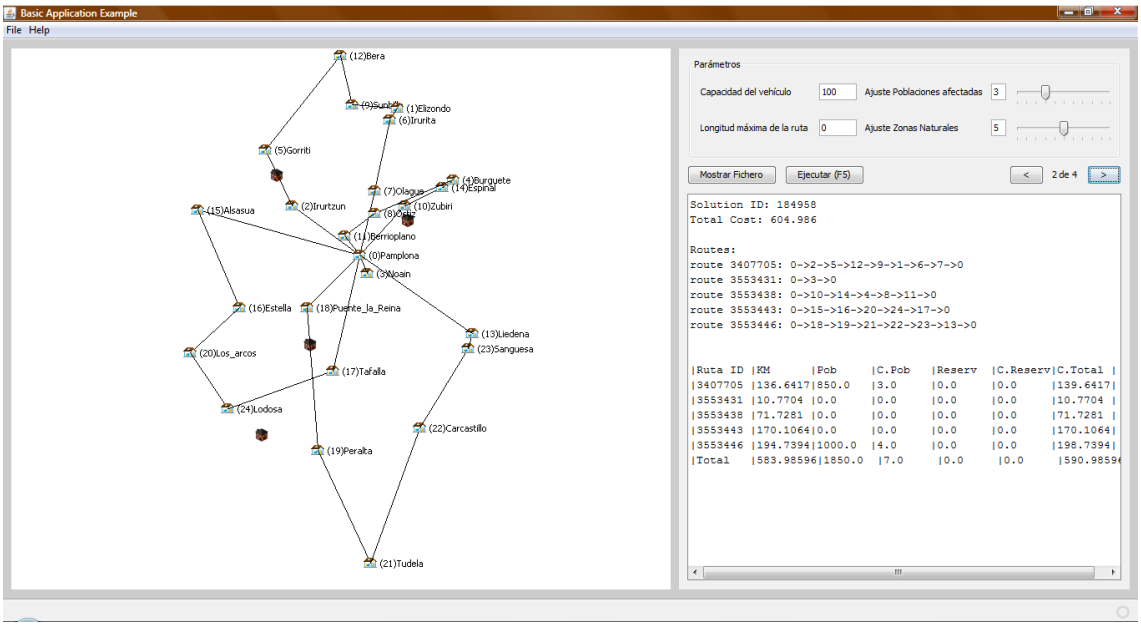


Imagen 5. 7

Como se puede ver en la imagen, la ruta atraviesa Larraga y Gorriti y el coste total de esta ruta es 604.986.

Coste total: 603.66

Coste kilométrico: 583.98

Coste Población: 7

Habitantes: 1850

Coste Reservas: 0

Kilómetros de reserva: 0

Obviamente, si nuestro objetivo es minimizar el coste y el número de personas afectadas por la ruta, escogeremos la primera solución, cuyo coste es de 603. 6657 y no atraviesa ninguna población.

5.2.3. Prueba 3. Clientes y Zonas naturales

En las pruebas de este apartado tendremos en cuenta los nodos de los clientes que forman el problema, así como las tres zonas naturales seleccionadas de Navarra.

Para el primer ejemplo, se utilizará el valor de ajuste de zonas naturales de 1.

El resultado es el siguiente:

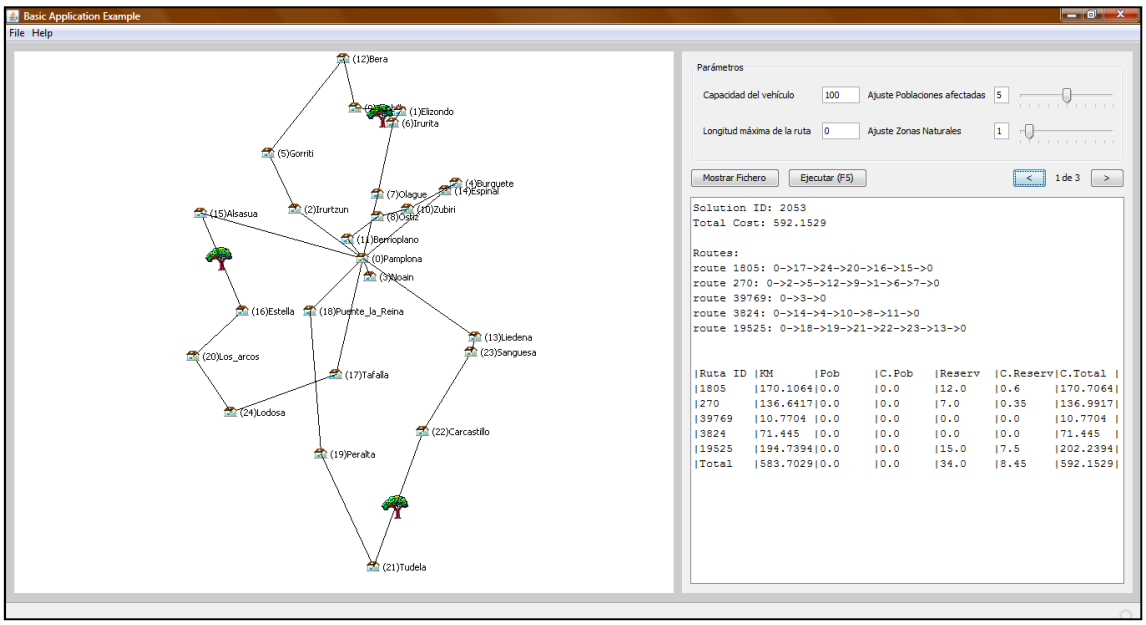


Imagen 5. 8

La mejor de las tres soluciones obtenidas, cuyo coste total es de 592.152, atraviesa las 3 zonas naturales del ejemplo.

Coste total: 592.15

Coste kilométrico: 583.70

Coste Población: 0

Habitantes: 0

Coste Reservas: 8.45

Kilómetros de reserva: 34

Si queremos dar más importancia a las zonas naturales, debemos incrementar el valor de ajuste. En el siguiente ejemplo este valor será 5.

En esta prueba, la mejor solución (imagen siguiente) tiene un coste de 608.79 y evita todas las zonas naturales.

Sin embargo, la siguiente solución ofrecida por el programa, consigue una solución de coste total de 609, atravesando 2 zonas naturales.

El coste de esta segunda solución es debido a que el hecho de atravesar dichas zonas ha repercutido en ella. Su coste kilométrico en cambio es menor que la primera solución.

Nótese que ninguna de las soluciones ofrecidas atraviesa las Bardenas Reales, ya que esta es la zona de mayor interés en el ejemplo diseñado.

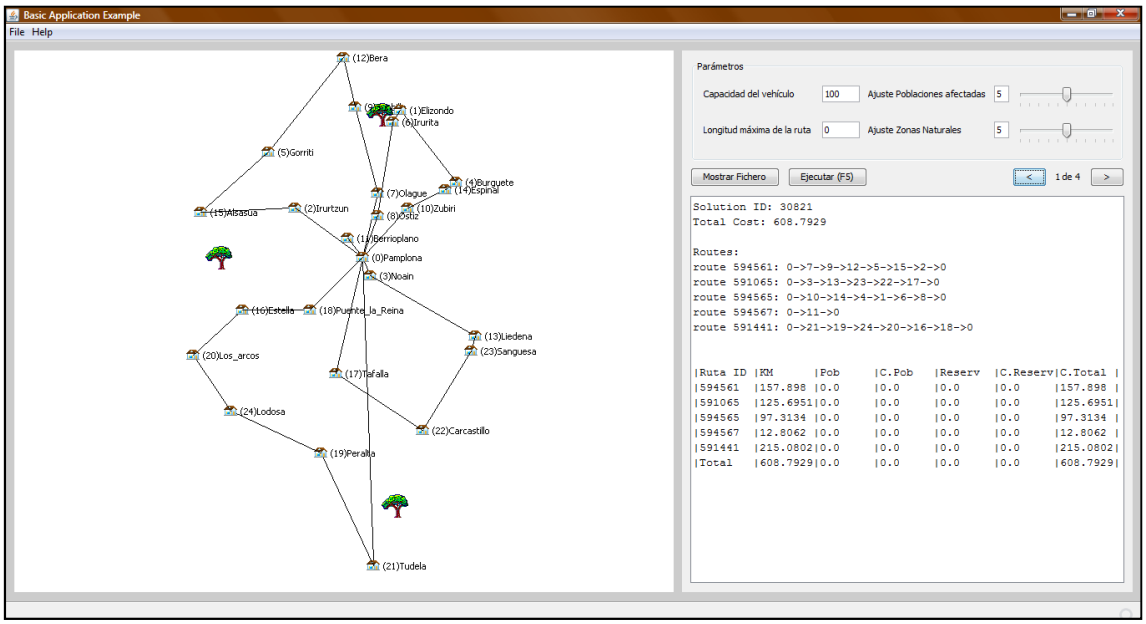


Imagen 5. 9

Coste total: 608.79

Coste kilométrico: 608.79

Coste Población: 0

Habitantes: 0

Coste Reservas: 0

Kilómetros de reserva: 0

Ya que la segunda solución tiene un coste kilométrico menor que la primera, y dado que la primera solución evita cruzar todas las zonas naturales, en lugar de incrementar el valor del ajuste de zonas naturales, lo que vamos a probar es a reducirlo. La siguiente prueba se realizará con el valor 3.

En este caso se puede observar que la mejor ruta propuesta por el GR-DSS es una en la cual se atraviesan 2 zonas naturales.

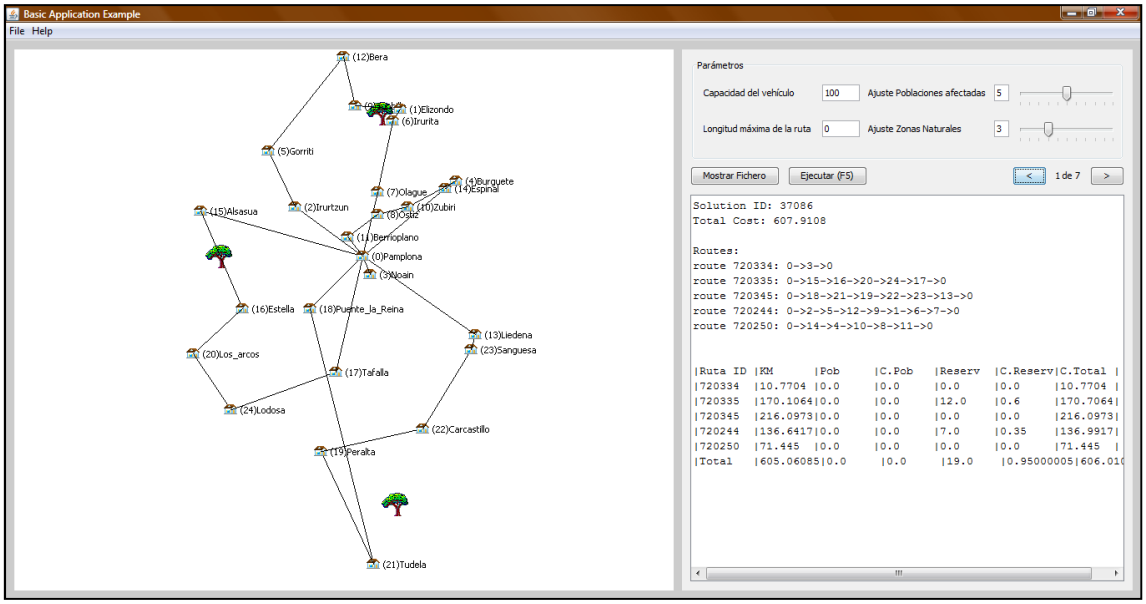


Imagen 5. 10

Ahora vemos que el coste total de esta ruta es de 607.9, frente a los 608.7 de la ruta que no atraviesa ninguna zona natural.

Coste total: 607.91

Coste kilométrico: 605.06

Coste Población: 0

Habitantes: 0

Coste Reservas: 0.95

Kilómetros de reserva: 19

5.2.4. Prueba 4. Clientes, poblaciones de paso y zonas naturales

En esta prueba el problema se complica, ya que el número de obstáculos crece y el algoritmo ha e combinar las 3 matrices de costes para realizar los cálculos.

Empezamos con valores de ajuste 1 y 1

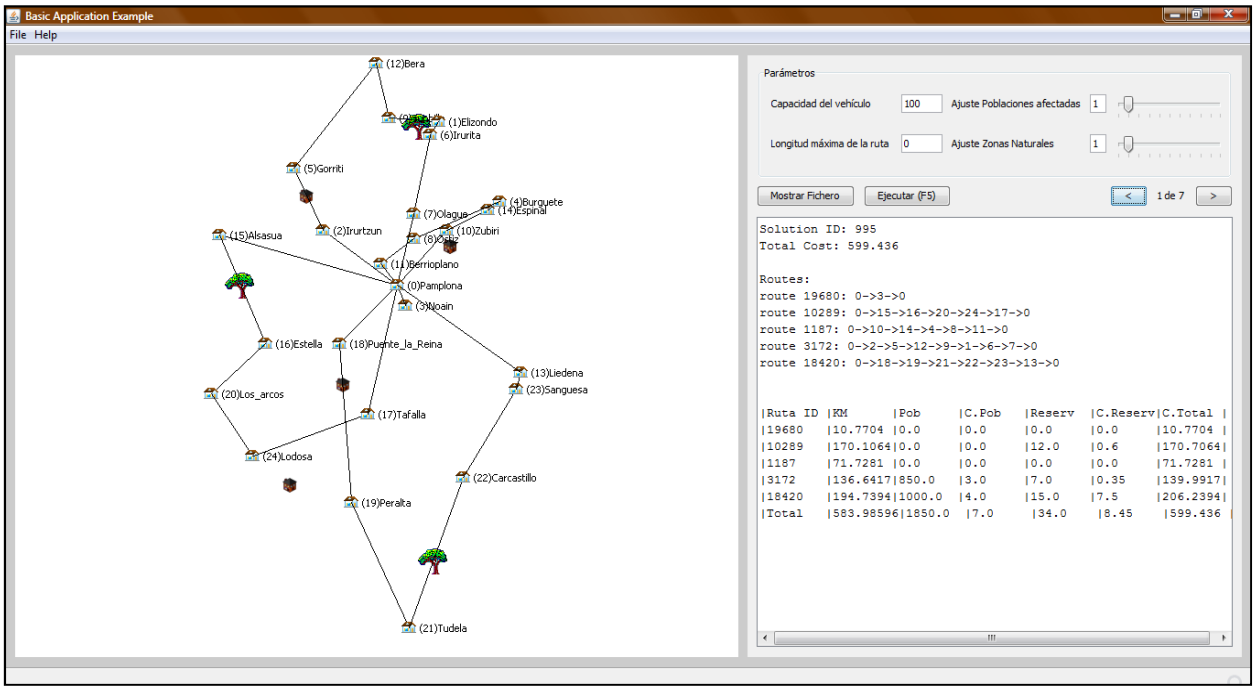


Imagen 5. 11

Coste total: 599.4

Coste kilométrico: 583

Coste Población: 7.0 (Larraga, Lekumberri)

Habitantes: 1850

Coste Reservas: 8.45 (Bértiz, Urbasa, Las Bardenas)

Kilómetros de reserva: 34

A continuación, hacemos la prueba con valores de ajuste: 1, 3

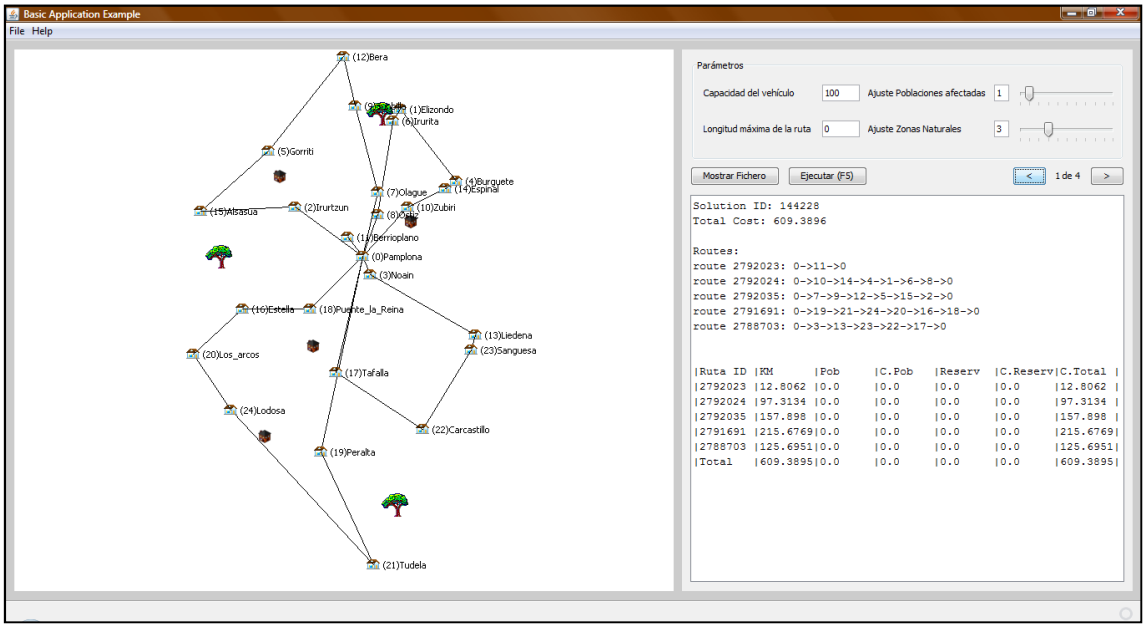


Imagen 5. 12

Los resultados obtenidos son los siguientes:

Coste total: 609.39

Coste kilométrico: 609.39

Coste Población: 0.0

Habitantes: 0.0

Coste Reservas: 0.0

Kilómetros de reserva: 0.0

Esta solución en concreto se dará como la mejor para alguna de las pruebas realizadas a continuación. En este caso la daremos como buena, pero para el resto de pruebas la obviaremos y estudiaremos las siguientes.

La razón es que esta solución es una de las que podríamos considerar como deseables. La totalidad de su coste está formado por los kilómetros recorridos, ya que no atraviesa ninguna zona natural ni tampoco ninguna de las poblaciones de paso.

Podemos decir que esta solución en concreto es, de entre todas las posibles alternativas que no cruzan ningún pueblo ni zona natural, la que menor coste tiene.

El diseño concreto del problema de Navarra ha hecho que en este caso, además de no atravesarse ninguno de los obstáculos, el coste kilométrico es bastante bajo. Esta es la razón de que esta solución salga en muchas de las pruebas.

Se aumenta el "peso" de las zonas naturales, los nuevos valores de ajuste: 1, 5

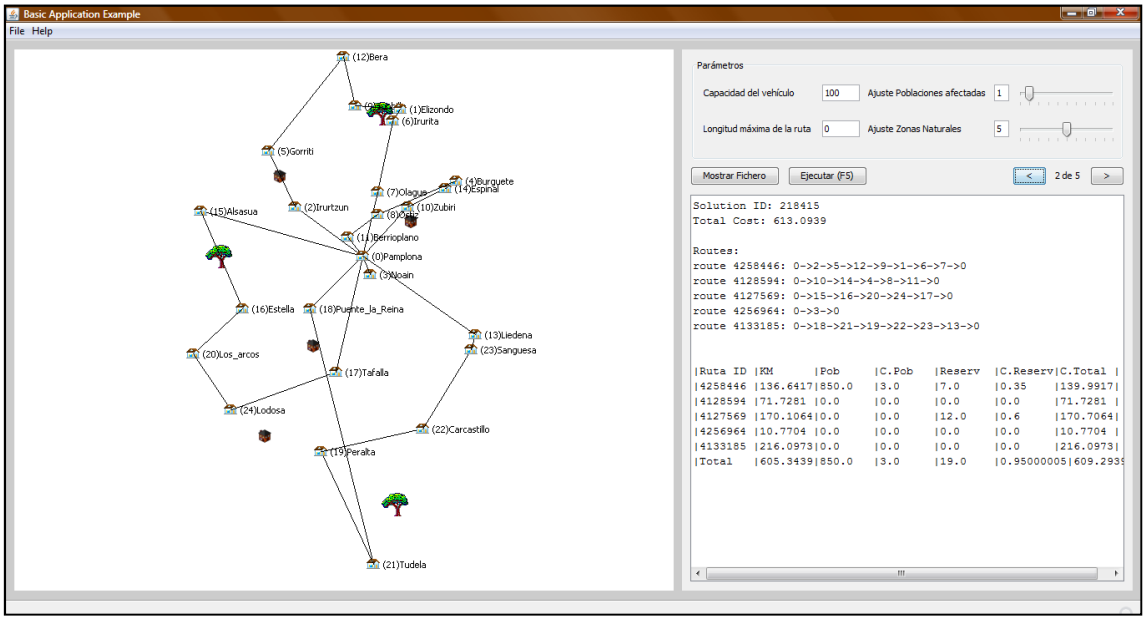


Imagen 5. 13

(Solución 2)

Coste total: 613.09

Coste kilométrico: 605.34

Coste Población: 3

Habitantes: 850

Coste Reservas: 0.95

Kilómetros de reserva: 19

En este caso, los valores de ajuste seleccionados son: 2, 1

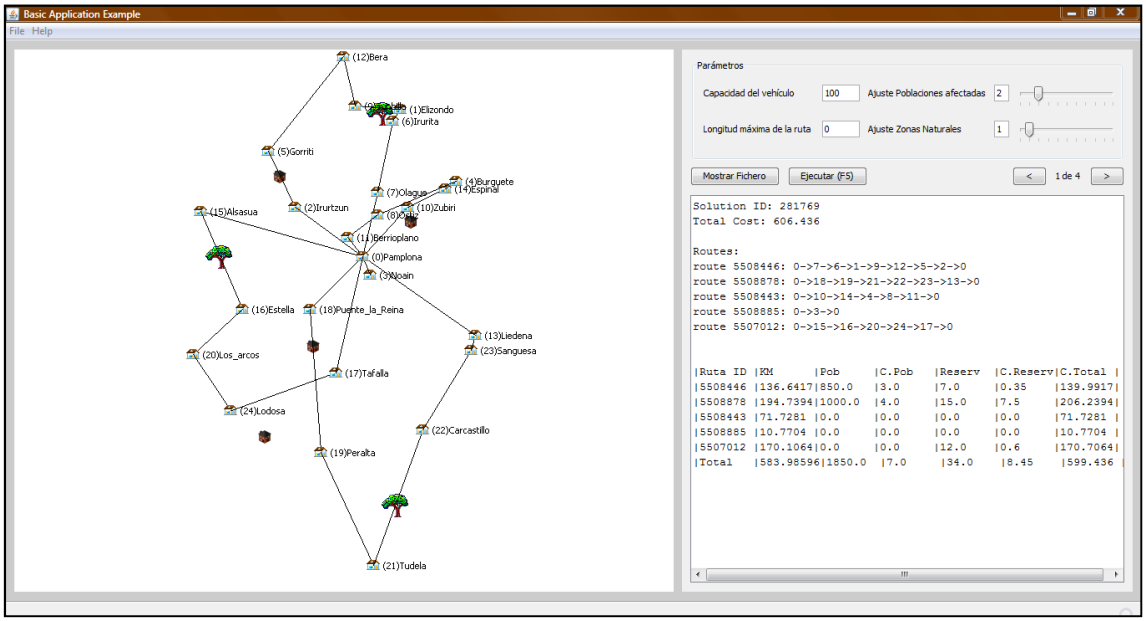


Imagen 5. 14

Coste total: 606.436

Coste kilométrico: 583.98

Coste Población: 7

Habitantes: 1850

Coste Reservas: 8.45

Kilómetros de reserva: 34

Para la siguiente prueba, valoraremos del mismo modo ambos criterios, para lo cual seleccionamos los valores de ajuste: 2, 2

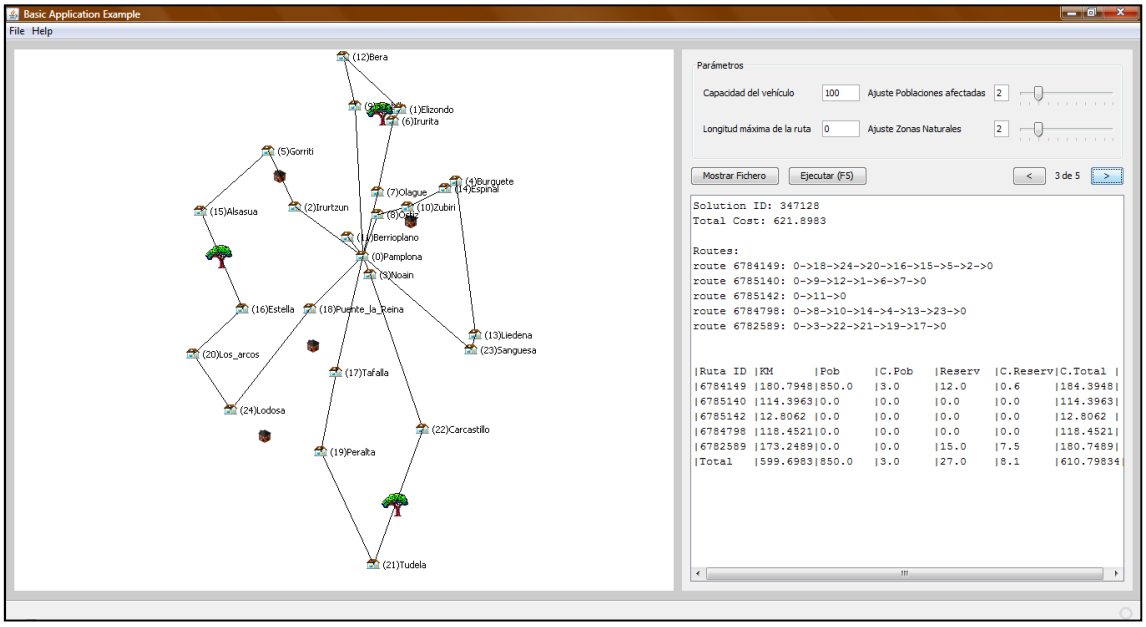


Imagen 5. 15

(Solución 3)

Coste total: 621.89

Coste kilométrico: 599.69

Coste Población: 3

Habitantes: 850

Coste Reservas: 8.1

Kilómetros de reserva: 27

Valores de ajuste: 2, 3

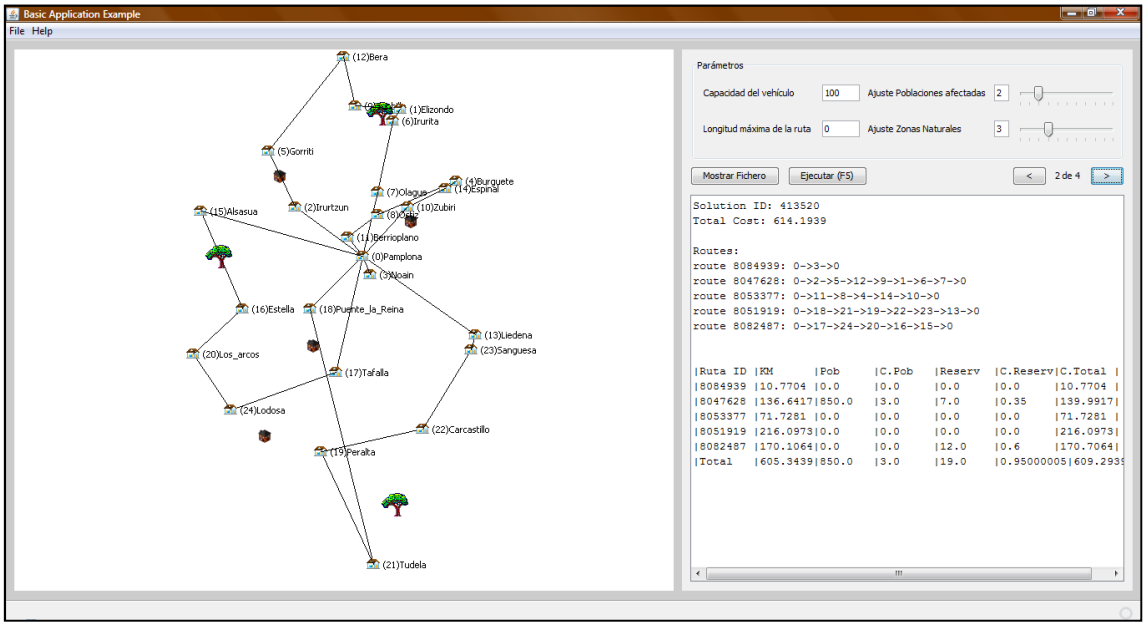


Imagen 5. 16

(Solución 2)

Coste total: 614.19

Coste kilométrico: 605.34

Coste Población: 3

Habitantes: 850

Coste Reservas: 0.95

Kilómetros de reserva: 19

Valores de ajuste: 3, 1

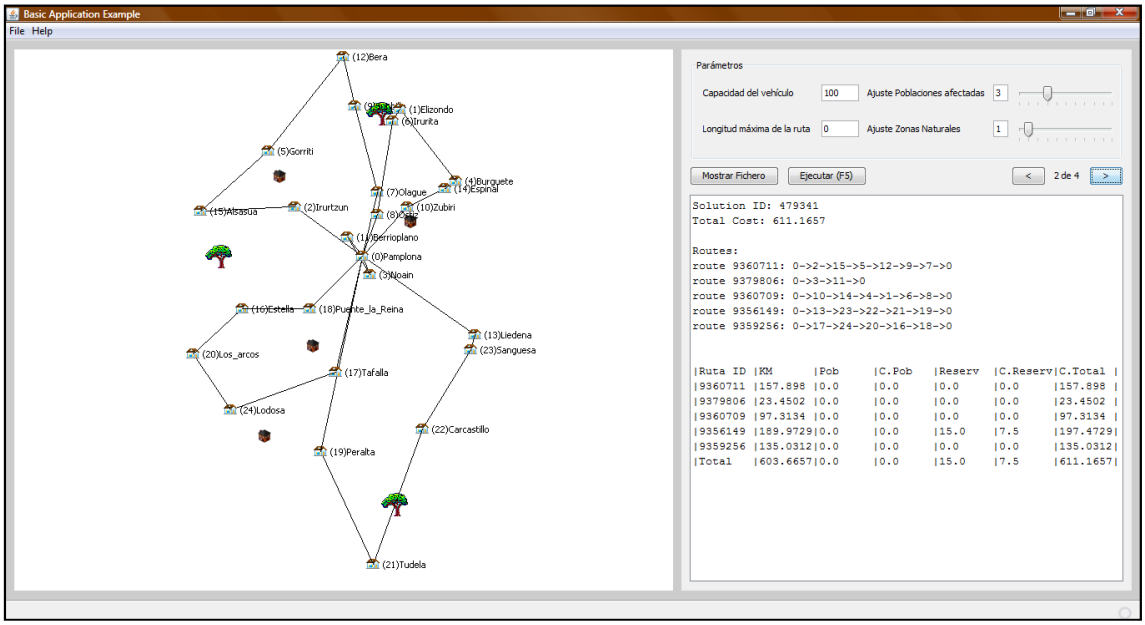


Imagen 5. 17

(Solución 2)

Coste total: 611.16

Coste kilométrico: 603.66

Coste Población: 0

Habitantes: 0

Coste Reservas: 7.5

Kilómetros de reserva: 15

Valores de ajuste: 3, 2

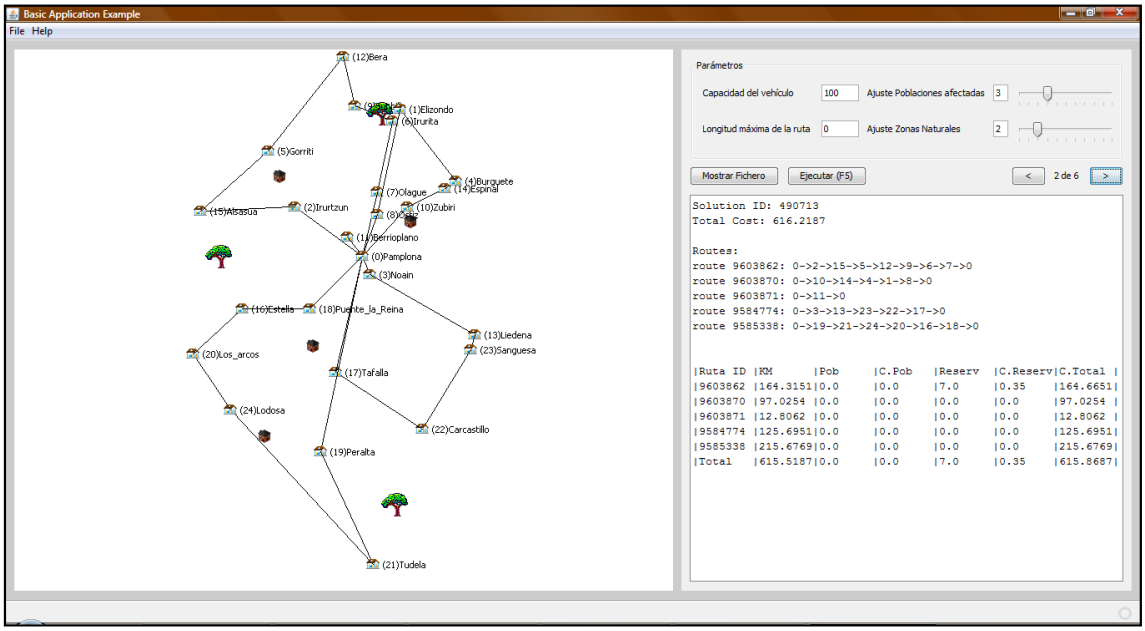


Imagen 5. 18

(Solución 2)

Coste total: 616.21

Coste kilométrico: 615.51

Coste Población: 0

Habitantes: 0

Coste Reservas: 7

Kilómetros de reserva: 0.35

Valores de ajuste: 3, 3

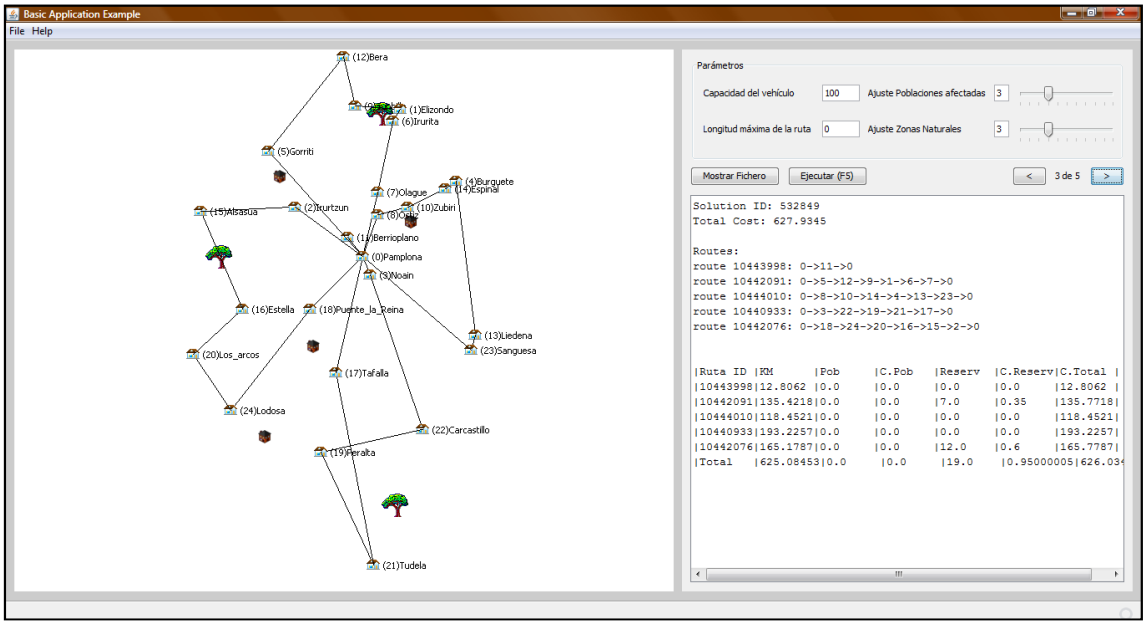


Imagen 5. 19

(Solución 3)

Coste total: 627.93

Coste kilométrico: 625.08

Coste Población: 0

Habitantes: 0

Coste Reservas: 19

Kilómetros de reserva: 0.95

Valores de ajuste: 10, 10

En este caso, vamos a “forzar” al programa para que intente buscar soluciones y que todas ellas eviten las zonas de paso y las zonas naturales.

Al poner ambos parámetros al máximo valor (10) el GR-DSS priorizará aquellas rutas en las que los costes de pueblos de paso y de zonas naturales no existan, ya que se ven claramente penalizados por el alto valor utilizado.

En este caso, creo que es relevante mostrar todas las soluciones obtenidas.

La primera es la misma que en el caso 1,3 y la que tomaremos como la ruta más recomendada por el GR-DSS.

La segunda solución mostrada es:

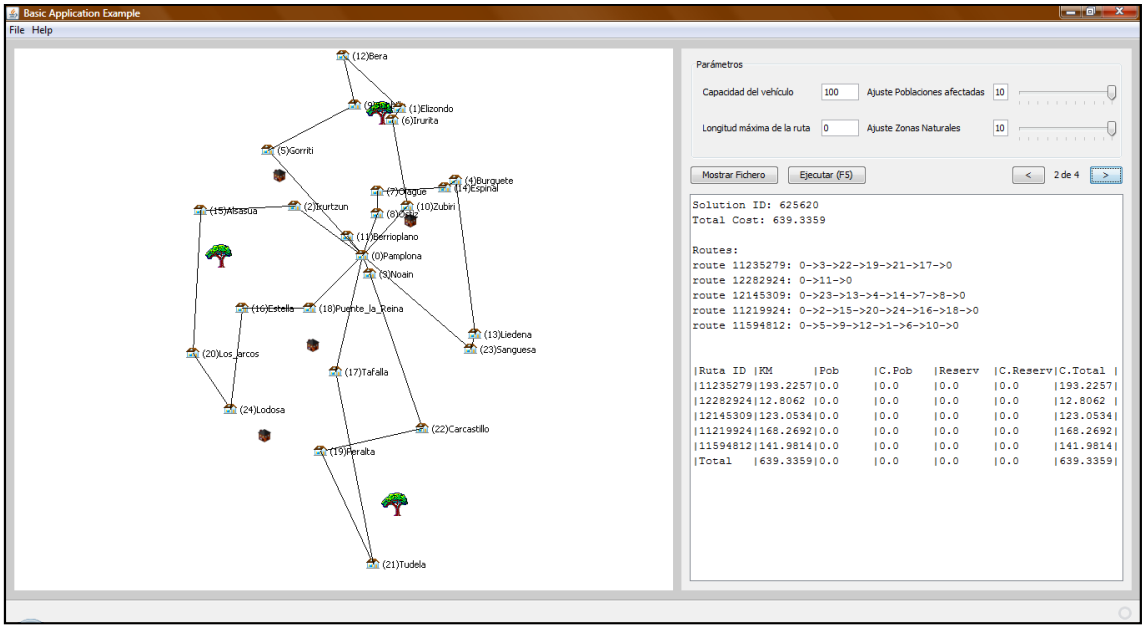


Imagen 5. 20

Coste total: 639.33

Coste kilométrico: 639.33

Coste Población: 0

Habitantes: 0

Coste Reservas: 0

Kilómetros de reserva: 0

La siguiente solución que encuentra el GR-DSS (valores 10, 10)es:

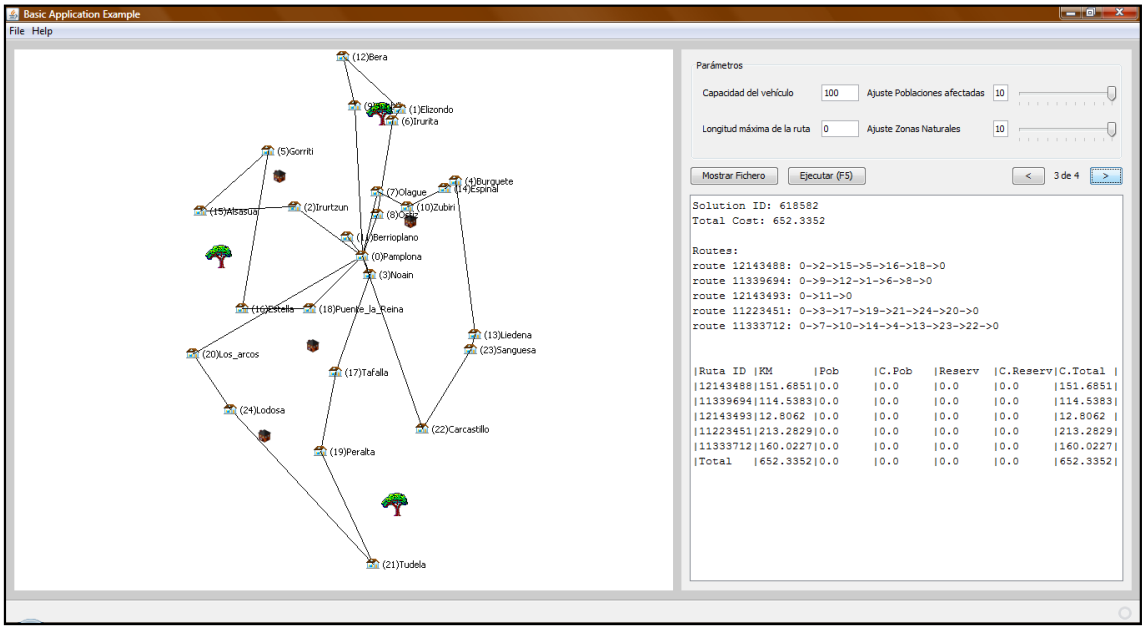


Imagen 5. 21

Coste total: 652.33

Coste kilométrico: 652.33

Coste Población: 0

Habitantes: 0

Coste Reservas: 0

Kilómetros de reserva: 0

Finalmente, la última solución conseguida (valores 10, 10):

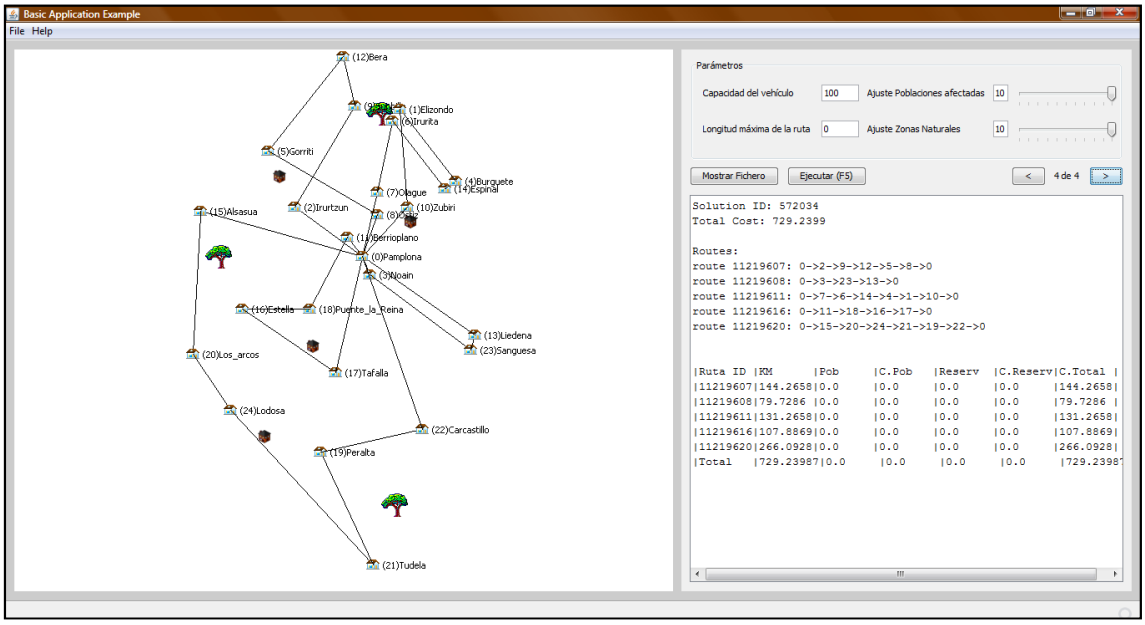


Imagen 5. 22

Coste total: 729.23

Coste kilométrico: 729.23

Coste Población: 0

Habitantes: 0

Coste Reservas: 0

Kilómetros de reserva: 0

Para ver el detalle de todas las soluciones generadas en las anteriores pruebas, véase el Anexo 2: Soluciones.

6. CONCLUSIONES Y MEJORAS FUTURAS

En este capítulo se resumen algunas de las ideas que han ido surgiendo durante el desarrollo de este proyecto de fin de carrera. Algunas de estas ideas no han podido llevarse a la práctica ya que sobrepasaban los plazos de tiempo y los recursos establecidos para dicho proyecto.

Sin embargo, pueden resultar de guía para futuros trabajos e investigaciones en este campo. Si bien el objetivo principal de este trabajo era presentar un proyecto de fin de carrera, la herramienta DSS que se ha desarrollado podría evolucionarse en un futuro, mejorando en muchos puntos, de manera que el resultado podría llegar a ser de utilidad para casos prácticos reales relacionados con el transporte de mercancías.

Esta reflexión no corresponde a ningún estudio del posible mercado de una herramienta informática de este tipo, sino que responde a la observación por parte del autor del estado actual del mundo del transporte y del camino que están tomando los estamentos gubernamentales para mitigar los efectos nocivos del mismo.

Como conclusión general, podemos decir que la herramienta GR-DSS se encuentra en una fase inicial de lo que se llama “ciclo de vida del software”, lo que quiere decir que aun quedaría un largo camino antes de lograr una aplicación informática de propósito comercial.

Sin embargo, gracias a esta primera versión del GR-DSS se sientan las bases de lo que puede ser una vía de trabajo para futuros estudiantes o investigadores en el campo de las aplicaciones informáticas para el cálculo de rutas.

Uno de los principales atractivos del programa, es que ha sido capaz de incorporar criterios medioambientales al proceso de cálculo de rutas, e integrarlo en un entorno de toma de decisiones. Sin duda, esta heterogénea combinación de factores, hace que la herramienta GR-DSS sea única, o al menos, no se han encontrado herramientas comerciales con tales intereses.

6.1. Conclusiones sobre el ejemplo de Navarra

Si nos fijamos en el ejemplo que se ha realizado para este proyecto podemos extraer una serie de conclusiones, muchas de las cuales son extensibles a cualquier problema que se intente ejecutar.

Las condiciones y características propias de cada problema son factores determinantes a la hora de encontrar rutas soluciones usando el GR-DSS para ello. En el caso preparado como ejemplo de Navarra (véase capítulo 4) se dan una serie de peculiaridades que hacen de este problema un caso especial.

Para empezar, gracias al GR-DSS se ha encontrado la mejor solución sin tener en cuenta ninguna restricción del tipo pueblos de paso o zonas naturales.

El coste de esta solución es de 583.7 Km, y este valor es el mismo para el coste total y para el coste kilométrico de la solución. Obviamente, los costes coinciden cuando no se tienen en cuenta las estricciones medioambientales. Solución obtenida en la prueba 1, capítulo 5.

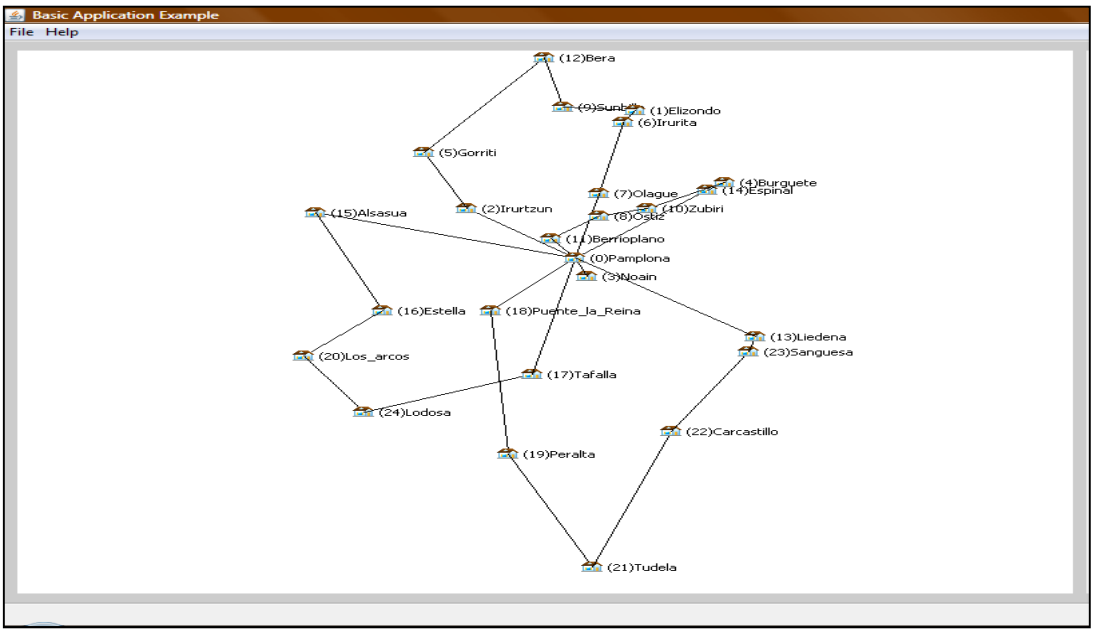


Imagen 6. 1

Este valor, 583.7, puede ser usado como referencia inicial, con la que poder comparar el resto de las soluciones obtenidas.

Debido a las características propias del problema y tras realizar varias pruebas con distintos ajustes, la conclusión a la que se llega es que la mejor solución general para este problema concreto, se obtiene fácilmente al cambiar los valores de ajuste al máximo posible, 10 y 10.

La solución que obtenemos de esta manera aparece en numerosas ocasiones como la mejor con diferentes combinaciones de los valores de ajuste, como por ejemplo con los ajustes 1 y 3, pero como referencia tomamos la obtenida con los ajustes 10 y 10.

La primera solución que se obtiene con los parámetros 10 y 10 es la siguiente:

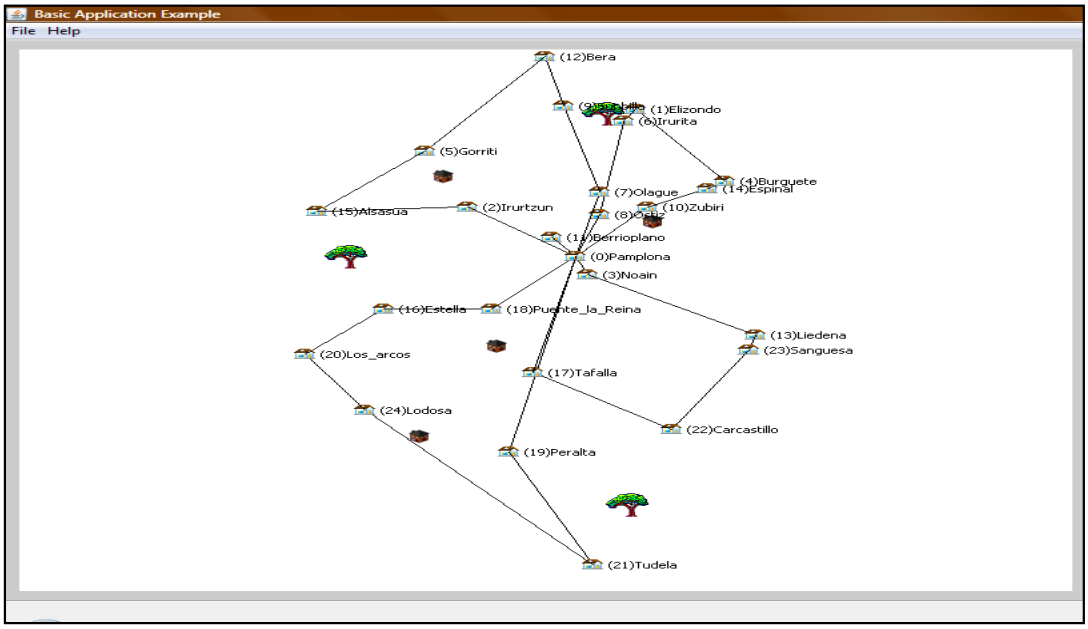


Imagen 6. 2

Si analizamos la imagen, es fácil observar que ninguna arista de ninguna subruta atraviesa ni pueblos de paso ni zonas naturales.

El coste total, que coincide con el kilométrico, es de 609.4 Km.

Si lo comparamos con la solución inicial (de 583.7 km) esta nueva supone un incremento del coste del 25.7 Km, lo cual supone un 4,4 % más respecto a la inicial.

Este incremento del 4,4%, que podría considerarse bajo-moderado, supone no atravesar ninguna de las zonas protegidas, ni pueblos de paso ni zonas naturales. Por lo tanto, ya que el objetivo es minimizar el impacto en las zonas seleccionadas, la ruta solución que mejor cumple este requisito es la de 609.4 Km.

En el resto de pruebas realizadas, con otros parámetros, se obtienen otras soluciones, algunas de ellas con un incremento del coste inferior, pero en todas ellas, al menos una de las zonas naturales o alguna de las poblaciones de paso son atravesadas.

Como se ha dicho anteriormente, el caso de la solución de 609.4 km se consigue debido a la peculiaridad del ejemplo en concreto. Si observamos las siguientes soluciones encontradas con los ajustes 10 y 10, se comprueba que los costes son mucho mayores.

Por ejemplo, la siguiente mejor solución que no atraviesa ninguna zona natural ni pueblo de paso, tiene un coste total de 639.3, es decir, un 9,5% más que la solución inicial.

La siguiente ruta que cumple estos requisitos tiene un coste de 652.3, por lo tanto, un 11,75% de incremento respecto a la solución inicial.

Por último, la siguiente mejor solución obtenida tiene un coste de 729.2, es decir, un 24,9 % más que la inicial.

Lo que se quiere explicar con esto es que es muy difícil encontrar soluciones que cumplan el requisito de no atravesar ningún punto protegido y que además tenga un incremento tan bajo como la de 609.4, con sólo un incremento del 4,4%.

En general, para problemas preparados para el GR-DSS no deberíamos encontrar tan fácilmente soluciones "tan buenas" como la anteriormente mencionada.

Por ello, en muchos resultados, la anterior solución se obvia y se muestran directamente las siguientes mejores soluciones.

Mejor Solución encontrada (para cada prueba)

En el capítulo 5, se explican todas las pruebas realizadas, con diferentes combinaciones de ajustes para las zonas naturales y las poblaciones de paso.

Junto a cada prueba se muestra la mejor de las soluciones encontradas, junto con un resumen de los diferentes costes de cada una de ellas.

El criterio que se ha usado para decidir cuál era "la mejor" de las soluciones, ha sido el del valor del Coste Total. Este mismo criterio es el que utiliza el GR-DSS para ordenar las rutas. Sin embargo, es necesario un estudio en profundidad para analizar exactamente cuál de las soluciones resuelve mejor las necesidades del usuario.

6.2. Mejoras futuras

Durante el diseño y desarrollo del GR-DSS han ido surgiendo multitud de ideas y mejoras posibles, muchas de las cuales no han podido llevarse a cabo por falta de tiempo y otras por la gran complejidad que implicaban.

A continuación se resumen algunas de ellas.

Interfaz

La interfaz gráfica debe ser uno de los puntos fuertes de un buen DSS. Por ello, una parte importante del tiempo invertido en el diseño y programación del GR-DSS se dedicó a mejorar la interfaz del programa.

Uno de los mayores problemas encontrados fue a la hora de escalar y centrar los problemas adecuadamente en el panel central.

El resultado final es bastante satisfactorio, pero creo que sería interesante trabajar en algunas mejoras.

Principalmente, los gráficos mostrados son muy básicos y en ocasiones no queda claro el sentido que siguen las rutas. En este punto, se podría intentar pintar las diferentes subrutas con diferentes colores y con flechas que indique el sentido de la ruta, en lugar de líneas simples.

Las imágenes que representan los nodos también pueden ser mejoradas. En este sentido, podría intentarse dibujar los nodos con un tamaño proporcional al número de habitantes de cada uno, ya que se posee esta información.

Por último, se pensó en crear una animación, de manera que aparezcan unos vehículos sobre el gráfico que recorran las diferentes subrutas obtenidas.

Información de salida y estadísticas

En cuanto a la información mostrada en pantalla, aunque el resultado es bastante completo en cuanto a los datos correspondientes a cada ruta, no se ha diseñado ningún método que permita comparar las diferentes rutas creadas.

Esto podría hacerse de manera gráfica o bien creando algún tipo de índice estadístico, que clasificase las soluciones según su idoneidad para encontrar soluciones.

Entrada de datos

En este apartado, puede decirse que la forma en la que están diseñadas las entradas de datos es un poco incómoda y demasiado simple.

Podría intentarse que, en lugar de crear ficheros de texto con los datos del problema, el programa fuera capaz de importar la información desde algún tipo de plataforma GIS (Geographic Information System) o desde alguna aplicación web como puede ser Google Maps.

En cuanto a la entrada de datos, tal y como está estructurado el programa actual, sería interesante la creación de un cuarto fichero de entrada que incluyese la información de las carreteras que no existen. De este modo evitaríamos uno de los inconvenientes que tiene trabajar con este tipo de algoritmos, que dan por hecho que existen aristas para cada par de nodos.

La forma que debería tener este algoritmo es similar a la de los ficheros de pueblos de paso y al de zonas naturales a evitar. Bastaría con "poner" un obstáculo con un coste enorme entre cada par de nodos que no están comunicados directamente.

De este modo se conseguiría unos resultados mucho más acordes con la realidad y sería más fácil trasladarlos a la práctica.

Salida de datos (google maps o gis)

Del mismo modo que las entradas, las salidas podrían exportarse a algún tipo de aplicación GIS o Google Maps.

Este punto podría ser uno de los principales objetivos de mejora. Una vez obtenida la solución para un problema concreto, puede ser muy interesante exportar los datos a una aplicación la cual sea capaz de dibujar dicha solución en un sistema de mapas de carreteras real.

De este modo, la salida gráfica sería mucho más completa, y principalmente, con información real de distancias, tipos de carreteras, peajes…

General

Podemos decir que el programa necesitaría una serie de revisiones para su refinamiento y ajuste. En este proyecto se presenta una primera versión, en la cual se pueden encontrar muchos fallos.

No es objetivo del proyecto crear una aplicación refinada y lista para ser comercializada, sino el empezar con un enfoque nuevo sobre el VRP clásico y combinarlo con una serie de restricciones de carácter medioambiental.

6.3. Posibles aplicaciones futuras

Hasta este momento se han explicado las capacidades del GR-DSS en cuanto al cálculo y diseño de rutas de transporte respetuosas con el medio y con la población.

Sin duda, con los criterios restrictivos impuestos en el programa, se consigue cumplir unos objetivos concretos en cuanto a sostenibilidad del transporte de mercancías por carretera.

Como ya se ha explicado, las diferentes leyes que afectan al ámbito del transporte cada vez son más restrictivas en cuanto a las emisiones y otros factores perjudiciales producidos por esta actividad, y se prevé que se continúe con esa misma tendencia.

En este sentido, la configuración actual del GR-DSS podría resultar interesante para todo tipo de empresas y particulares que tuvieran que pagar algún tipo de arancel como contraprestación por el tránsito a través de alguna zona sensible al transporte.

En la actualidad, aun no existen este tipo de impuestos al transporte, pero como se ha informado en el capítulo 3, estas medidas entran dentro de los planes a medio plazo de la UE.

Cabe destacar el hecho de que el GR-DSS puede ser modificado y adaptado a otros muchos tipos de situaciones relacionadas con el cálculo de rutas con algún tipo de restricción.

Hasta ahora, existen múltiples herramientas dedicadas al cálculo de rutas, pero, por lo general no posibilitan la inclusión de restricciones en las mismas, más allá de evitar peajes o similares.

Con futuras modificaciones, el GR-DSS podría incluir otro tipo de criterios, relacionados o no con la sostenibilidad del transporte. Por ejemplo, podría incluirse restricciones a las rutas de modo que se evitasen ciudades, pueblos, peajes… pero aun podría irse más lejos e incluir criterios más especializados como podría ser las siguientes propuestas:

Evitar zonas con pendientes pronunciadas: para determinado tipo de cargas (gran volumen) y vehículos, es posible que determinadas pendientes impidan la circulación o generen algún tipo de problema. Se podría buscar rutas alternativas, con menos pendientes y por tanto más adecuadas.

Calcular rutas que eviten radares fijos: Conociendo la ubicación exacta de los radares (información publicada por la DGT), podemos calcular rutas que eviten dichas zonas.

Del mismo modo, podría evitarse el tránsito por zonas clasificadas como puntos negros o de especial peligro.

También puede modificarse el software, de modo que sea utilizado para moverse dentro de una gran ciudad. En este punto, podríamos evitar pasar por zonas residenciales, por parques, cerca de colegios, hospitales, residencias…etc.

En este proyecto se estudia un ejemplo concreto para el ámbito de Navarra, sin embargo, los problemas admitidos por el programa no quedan limitados a un espacio regional, sino que puede admitir y tratar problemas a una escala mucho mayor, por ejemplo, para calcular rutas a nivel nacional o europeo.

GR-DSS es un prototipo de DSS con criterios medioambientales, pero como se ha dicho, las posibilidades que puede ofrecer esta filosofía (cálculo de rutas con restricciones de paso) son enormes y, su aplicación a la práctica de empresas y particulares puede ser de gran interés en el futuro.

Bibliografía

Ballou R. H., Handoko Rahardja, Noriaki Sakai. *Selected country circuity factors for road travel distance estimation.* Transportation Research Part A 36 (2002) 843–848

Ballou, R.H. (1999): *Business Logistics Management. Planning, Organizing, and Controlling the Supply Chain*. Prentice Hall, Upper Saddle River, New Jersey.

Barrios González, C., Martínez Navarro, M.A., Sánchez de Molina Martín, J., Toribio Muñoz. M.R., (2003). *La tarificación viaria y sus efectos sobre el bienestar social. Estudios de economía aplicada,* Vol. 21 – 2.

Clarke, G., Wright, J.W. (1964): "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Operations Research*, Vol. 12.

Davenport, Tom. *Decision Evolution*. Artículo digital consultado en http://www.cio.com/archive/100104/keynote.html

Embleton, T. F. W., (1996). *Tutorial on sound propagation outdoors. Journal of Acoustic. Society.* Am. 100 (1), July 1996.

Erkut E. y Verter V (1997): *MODELING OF TRANSPORT RISK FOR HAZARDOUS MATERIALS.* Operations Research, Vol. 46, No. 5, September–October 1998.

F. Burstein, C.W. Holsapple. *Hanbook On decision Support Systems.* Vol. 1. International handbooks on Information Systems. Springer, 2008.

Faulín J. and Juan A. (2008): "*The ALGACEA-1 Method for the Capacitated Vehicle Routing Problem*". International Transactions in Operational Research.

Feo T.A. and M.G.C. Resende (1995) *Greedy randomized adaptive search procedures*. Journals of Global Optimization, 6:109–133, 1995.

Foro medioambiental del transporte por carretera. (2003): "*La gestión medioambiental en organizaciones de transporte*". Fundación Francisco Corell.

Golden B, Raghavan S and Wasil EE (eds.) (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer. New York, USA

INFRAS (2006): "*Costes externos del transporte en el País Vasco*". Informe final. Leioa/Zurich.

Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazález, Alberto Larzabal, Jesús Calleja, Jon García (2004). *Aprenda Java como si estuviera en Primero.*

Juan Angel A., Faulín J, Ruiz Rubén, Barrios Barry, Caballé Santi. (2009) *The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem.*

Juan Angel A., Faulín J., Jorba J., D Riera, D Masip and B Barrios (2008). *On the Use of Monte Carlo Simulation, Cache and Splitting Techniques to Improve the Clarke and Wright Savings Heuristics*

Mauricio G. C. Resende. *Metaheuristic Hybridization with Greedy Randomized Adaptive Search Procedures*. INFORMS 2008

Nolan, Richard. *"Managing the computer resource: a stage hypothesis"*. (1973) Communications of the ACM (Association for Computing Machinery)

O'Brien, J. (2003): *Introduction to Information Systems*. Essentials for the e-Business Enterprise. McGraw-Hill. Capítulo 9.

Observatorio hispano-frances del tráfico en los Pirineos. (2008) "*Documento nº5. Diciembre 2008"*. Ministerio de Fomento, Gobierno de España. Ministère de l'Equipement des Transports du Logement du Tourisme et de la Mer, Gouvernement Française.

Toth P. and Vigo D. (2002): *THE VEHICLE ROUTING PROBLEM*. Society for Industrial & Applied Mathematics, Philadelphia.

Toth P. and Vigo D. (2003). *The Granular Tabu Search and its Application to the Vehicle Routing Problem*. INFORMS Journal on Computing.

Turban, E., Aronson, J., Liang, T.P. *Decision Support Systems and Intelligent Systems. Prentice Hall, 2005. Seventh edition*. 0-13-046106-7

Recursos web consultados

http://neo.lcc.uma.es/radi-aeb/WebVRP/ *The VRP web*

http://preguntaslinux.org/tutorial-de-netbeans-visual-library-t-4658.html

http://www.eea.europa.eu/es/themes/transport

http://www.greenpeace.org/espana

http://www.scribd.com/doc/967380/Tutorial-Netbeans

http://www.transpirenaica.org

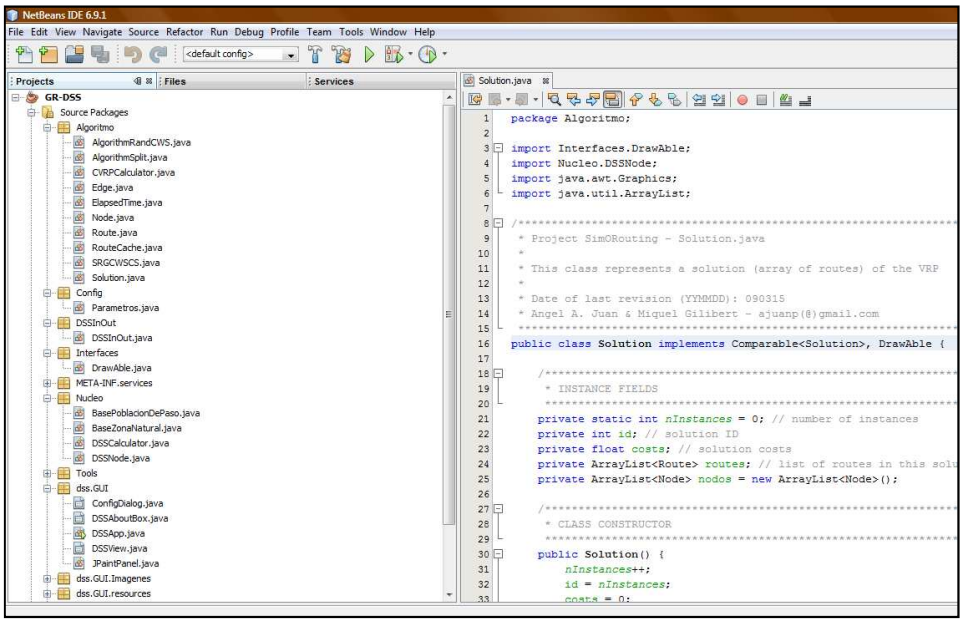
http://www.WWF.es

Anexo 1. Código completo del GR-DSS

Organización de las clases del programa

Como ya se ha comentado, el GR-DSS ha sido programado en el lenguaje JAVA en el entorno de programación de *Sun Microsystems*, NetBeans IDE 6.9.1.

Al tratarse de un lenguaje orientado a objetos, el código se estructura en clases y paquetes. A continuación una vista de NetBeans de la jerarquía de clases y paquetes.



A continuación, se expone el código del programa, clasificado por paquetes y clases.

En el paquete “algoritmo” se encuentran las clases principales del algoritmo original SimuRoute diseñado por (C) Angel A. Juan - ajuanp(@)gmail.com.

En esta versión, algunas partes del anterior algoritmo han sido modificadas para su correcto ensamblado con el resto del programa.

El resto de las clases de los demás paquetes son originales creadas por el autor de este proyecto de fin de carrera.

Paquete “Algoritmo”

Clase AlgorithmRandCWS

```
package Algoritmo;

import Interfaces.DrawAble;
import Nucleo.DSSNode;
import java.awt.Graphics;
import java.util.ArrayList;

/*****************************************************************************
 * Project SimORouting - Solution.java
 *
 * This class represents a solution (array of routes) of the VRP
 *
 *****************************************************************************/
public class Solution implements Comparable<Solution>, DrawAble {

   /*****************************************************************************
    * INSTANCE FIELDS
    *****************************************************************************/
   private static int nInstances = 0; // number of instances
   private int id; // solution ID
   private float costs; // solution costs
   private ArrayList<Route> routes; // list of routes in this solution
   private ArrayList<Node> nodos = new ArrayList<Node>();

   /*****************************************************************************
    * CLASS CONSTRUCTOR
    *****************************************************************************/
   public Solution() {
      nInstances++;
      id = nInstances;
      costs = 0;
      routes = new ArrayList<Route>();
   }

   /*****************************************************************************
    *  PUBLIC METHOD addRoute()
    *****************************************************************************/
   public void addRoute(Route aRoute) {
      routes.add(aRoute);
   }

   /*****************************************************************************
    *  PUBLIC METHOD addCosts()
    *****************************************************************************/
   public void addCosts(Route aRoute) {
      costs += aRoute.getCosts();
   }

   /*****************************************************************************
    *  PUBLIC METHOD setCosts()
    *****************************************************************************/
   public void setCosts(float c) {
      costs = c;
   }

   /*****************************************************************************
    *  PUBLIC METHOD getId()
    *****************************************************************************/
   public int getId() {
      return id;
   }

   /*****************************************************************************
    *  PUBLIC METHOD getCosts()
```

```
 ***************************************************************************/
public float getCosts() {
   return costs;
}

/***************************************************************************
 *  PUBLIC METHOD getRoutes()
 ***************************************************************************/
public ArrayList<Route> getRoutes() {
   return routes;
}

/***************************************************************************
 *  PUBLIC METHOD getDemand()
 ***************************************************************************/
public float getDemand() {
   float demand = 0;
   for (Route aRoute : this.routes) {
      demand = demand + aRoute.getDemand();
   }

   return demand;
}

public void setNodos(ArrayList<Node> nodos) {
   this.nodos = nodos;
}

/***************************************************************************
 *  PUBLIC METHOD getNodes()
 ***************************************************************************/
public ArrayList<Node> getNodes() {
   if (this.routes.isEmpty()) {
      return nodos;
   } else {
      ArrayList<Node> nodes = new ArrayList<Node>();
      ArrayList<Node> candidates = new ArrayList<Node>();

      // In any list of nodes, the first one is the depot
      for (int i = 0; i < routes.size(); i++) {
         Route iRoute = routes.get(i);
         candidates = iRoute.getNodes();
         for (int j = 0; j < candidates.size(); j++) {
            Node jCandidate = candidates.get(j);
            if (nodes.contains(jCandidate) == false) {
               if (jCandidate.getId() == 0) // the depot must be the first
               {
                  nodes.add(0, jCandidate);
               } else {
                  nodes.add(jCandidate);
               }
            }
         }
      }

      return nodes;
   }
}



/***************************************************************************
 *  PUBLIC METHOD toString()
 ***************************************************************************/
@Override
public String toString() {
   Route aRoute; // auxiliary Route variable
   String s = "";
   for (int i = 1; i <= routes.size(); i++) {
```

```
            aRoute = routes.get(i - 1);
            for (Edge e : aRoute.getEdges()) {
               if (e.getOrigin().getId() == 0) {
                  s = s.concat("0" + " " + e.getEnd().getId());
               } else {
                  s = s.concat(" " + e.getEnd().getId());
               }

            }
            s = s + "\r\n";
         }
         return s;
      }

      /*****************************************************************************
       *  PUBLIC METHOD compareTo()
       *****************************************************************************/
      public int compareTo(Solution otherSol) {
         Solution other = otherSol;
         float e1 = this.getCosts();
         float e2 = other.getCosts();
         if (e1 < e2) {
            return -1;
         } else if (e1 > e2) {
            return 1;
         }
         return 0;
      }

      public void draw(Graphics g, int AlturaPanel, int AnchuraPanel, float Escala, int DesplazamientoX, int DesplazamientoY, int
MaxXEscalada, int MaxYEscalada) {
         if (routes.isEmpty()) {
            for (Node n : nodos) {
               DSSNode nodo = (DSSNode) n;
               nodo.draw(g, AlturaPanel, AnchuraPanel, Escala, DesplazamientoX,
                     DesplazamientoY, MaxXEscalada, MaxYEscalada);
            }
         } else {
            for (Route r : routes) {
               r.draw(g, AlturaPanel, AnchuraPanel, Escala, DesplazamientoX,
                     DesplazamientoY, MaxXEscalada, MaxYEscalada);
             }
          }
      }
   }
```

Clase AlgorithmSplit

```
package Algoritmo;

import java.util.ArrayList;
import java.util.Random;

import umontreal.iro.lecuyer.randvar.RandomVariateGen;

/*****************************************************************************
 * Project SimORouting - AlgorithmSplit.java
 *
 * This class encapsulates the splitting methodology.
*****************************************************************************/

public class AlgorithmSplit
{
        /*****************************************************************************
   * INSTANCE FIELDS
   * ***************************************************************************/

        private int firstPolicy = 17; // First splitting policy
        private int lastPolicy = 57; // Last splitting policy
        private ArrayList <Edge> savingsList; // Original savingsList
        private AlgorithmRandCWS splitRCWS; // new instance of the RandCWS algorithm

        private float[] vrpCenter; // (x-bar, y-bar) is a geometric center for the VRP
        private float vrpCenterX; // x-bar = mean of x[i]
        private float vrpCenterY; // y-bar = mean of y[i]

        private ArrayList<Route> frontRoutes; // list of routes satisfying policy
        private ArrayList<Route> backRoutes; // routes not satisfying split policy
        private ArrayList<Node> frontNodes; // nodes in frontRoutes
        private Solution frontSubSol; // sub-solution derived from the frontRoutes
        private Solution backSubSol; // vrpSol = union(frontSubSol, backSubSol)


  /*****************************************************************************
   * CLASS CONSTRUCTOR
   * ***************************************************************************/

        public AlgorithmSplit(ArrayList<Node> vrpNodes, long[][] cMatrix,
                ArrayList<Edge> sList, float vCap, int maxCosts, int sCosts, Random rngJ,
                RandomVariateGen rngL, boolean useL, float bMin, float bMax )
        {
                // Set some instance fields
                savingsList = sList;
                splitRCWS = new AlgorithmRandCWS(cMatrix, vCap, maxCosts, sCosts,
                                rngJ, rngL, useL, bMin, bMax);


                /*****************************************************************************
             * GET THE (X-BAR, Y-BAR) COORDINATES FOR A CENTRAL VRP POINT
             *****************************************************************************/

                vrpCenter = calcGeometricCenter(vrpNodes);
        vrpCenterX = vrpCenter[0]; // x-bar coordinate
                vrpCenterY = vrpCenter[1]; // y-bar coordinate
        }


         /*****************************************************************************
   * PUBLIC METHOD improveSolUsingSplitting()
   *****************************************************************************/

  public Solution improveSolUsingSplitting(Solution vrpSol, int nIterPerSplit,
                RouteCache rc)
  {
```

```
// 1. Initialize dynamic currentVRPSol variable
Solution currentVRPSol = vrpSol;

// 2. For each possible splitting policy, split and solve
for ( int policy = firstPolicy; policy <= lastPolicy; policy++ )
{
        // 2.1. Construct a list with all routes in the currentVRPSol;
        ArrayList<Route> allRoutes = currentVRPSol.getRoutes();

        // 2.2. Set a geometric center for each route in the list
        setRoutesGeometricCenters(allRoutes);

        // 2.3. Split allRoutes in two subsets,
        // (frontRoutes will be the ones we will try to improve next, but
        //  we also need to know which are the complementary backRoutes)
        frontRoutes = calcFrontRoutes(allRoutes, policy);
        backRoutes = calcBackRoutes(frontRoutes, allRoutes);

        // 2.4. Continue this process only if there are enough routes
        if ( frontRoutes.size() > 1 && backRoutes.size() > 0 )
        {
                // 2.5. Construct the initial complementary sub-solutions
                frontSubSol = calcSubSol(frontRoutes);
                backSubSol = calcSubSol(backRoutes);

                // 2.6. Get lists of nodes associated with the selected routes
                frontNodes = frontSubSol.getNodes();

                // 2.7. Calculate splitSavingsList from original savingsList
                //  (notice that the resulting sublist will be already sorted)
                ArrayList<Edge> splitSavingsListF = new ArrayList<Edge>();
                for ( int i = 0; i < savingsList.size(); i++ )
                {
                        Edge iEdge = savingsList.get(i);
                        Node iEdgeOrigin = iEdge.getOrigin();
                        Node iEdgeEnd = iEdge.getEnd();
                        // Construct frontNodes list
                        if ( iEdgeOrigin.getId() < iEdgeEnd.getId() &&
                                        frontNodes.contains(iEdgeOrigin) &&
                                        frontNodes.contains(iEdgeEnd) )
                                splitSavingsListF.add(iEdge);
                }

                // 2.8. Start a random search process to improve frontSubSol
                int nIterF = nIterPerSplit; // * frontRoutes.size();
                for ( int i = 1; i <= nIterF; i++ )
                {
                        // 2.8.1. Get a random solution based on the Random CWS
                        Solution newSubSolF = splitRCWS.constructRandomSol(
                                        frontNodes, splitSavingsListF, true);

                        // 2.8.2. Apply the hash-table improvement and update sol
                        // (this can also help to quickly improve cached values)
                        newSubSolF = rc.improveRoutesUsingHashTable(newSubSolF);
                        newSubSolF = rc.improveAreaUsingHashTable(newSubSolF);

                        // 2.8.3. If appropriate, update frontSubSol
                        if ( newSubSolF.getCosts() < frontSubSol.getCosts() )
                                frontSubSol = newSubSolF;
                }

                // 2.9. Try a quick improvement of backSubSol
                //  (notice that improveRoutesUsingHashTable has been already
                //   applied to this set of routes in the SRGCWSCS.java)
                // backSubSol = rc.improveRoutesUsingHashTable(backSubSol);
                backSubSol = rc.improveAreaUsingHashTable(backSubSol);

                // 2.10. Re-construct the full VRP solution by using frontSubSol
                Solution newSol = unifySubSols(frontSubSol, backSubSol);
```

```
                        // 2.11. If appropriate, update currentVRPSol
                        if ( newSol.getCosts() < currentVRPSol.getCosts() &&
                                newSol.getNodes().size() == currentVRPSol.getNodes().size() &&
                                newSol.getDemand() - currentVRPSol.getDemand() <=
                                        SRGCWSCS.EPSILON )
                                currentVRPSol = newSol;
                }
        }

        // 3. Return the best full VRP solution found so far
        return currentVRPSol;
}


        /*********************************************************************************
         * PRIVATE METHOD setRoutesGeometricCenters()
         * Given a list of routes, it assigns to each route a geometric center
         *********************************************************************************/

        private void setRoutesGeometricCenters(ArrayList<Route> routes)
        {
                for ( int i = 0; i < routes.size(); i++ )
        {
                Route iRoute = routes.get(i);
                ArrayList<Node> iRouteNodes = iRoute.getNodes();
                float[] iRouteCenter = calcGeometricCenter(iRouteNodes);
                iRoute.setXRouteCenter(iRouteCenter[0]);
                iRoute.setYRouteCenter(iRouteCenter[1]);
        }
        }


        /*********************************************************************************
         * PRIVATE METHOD calcGeometricCenter()
         * Returns a geometric center for a set of nodes
         *********************************************************************************/

        private float[] calcGeometricCenter(ArrayList<Node> nodes)
        {
                // 1. Declare and initialize variables
                float sumX = (float) 0.0; // sum of x[i]
                float sumY = (float) 0.0; // sum of y[i]
                float[] center = new float[2]; // center as (x, y) coordinates

                // 2. Calculate sums of x[i] and y[i] for all iNodes in nodes
                Node iNode; // iNode = ( x[i], y[i] )
                for (int i = 0; i < nodes.size(); i++)
                {
                        iNode = nodes.get(i);
                        sumX = sumX + iNode.getX();
                        sumY = sumY + iNode.getY();
                }

                // 3. Calculate means for x[i] and y[i]
                center[0] = sumX / nodes.size(); // mean for x[i]
                center[1] = sumY / nodes.size(); // mean for y[i]

                // 4. Return center as (x-bar, y-bar)
                return center;
        }


        /*********************************************************************************
         * PRIVATE METHOD calcFrontRoutes()
         * Returns those routes from allRoutes that satisfy a given policy
         *********************************************************************************/

         private ArrayList<Route> calcFrontRoutes(ArrayList<Route> allRoutes,
```

```
                    int policy)
{
        ArrayList<Route> fRoutes = new ArrayList<Route>();
        double mPId8 = Math.tan(Math.PI / 8);
        double m3PId8 = Math.tan(3 * Math.PI / 8);

        switch (policy)
        {
                // POLICIES 1 TO 16: COVERING 270
                case 1: // From 180 to 90 (South & East)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                                        vrpCenterX <= allRoutes.get(i).getXRouteCenter() )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 2: // From 157.5 to 67.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 3: // From 135 to 45
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 4: // From 112.5 to 22.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 5: // From 90 to 0 (West & South)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                                        vrpCenterX >= allRoutes.get(i).getXRouteCenter() )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 6: // From 67.5 to 337.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 7: // From 45 to 315
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 8: // From 22.5 to 292.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
```

```
case 9: // From 0 to 270 (North & West)
        for (int i = 0; i < allRoutes.size(); i++ )
                if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                        vrpCenterX >= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
        break;
case 10: // From 337.5 to 247.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 11: // From 315 to 225
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 12: // From 292.5 to 202.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 13: // From 270 to 180 (North & East)
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                        vrpCenterX <= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
        break;
case 14: // From 247.5 to 157.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 15: // From 225 to 135
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 16: // From 202.5 to 112.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
// POLICIES 17 TO 32: COVERING 225
case 17: // From 180 to 45
        for ( int i = 0; i < allRoutes.size(); i++ )
                if(  vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                         allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 18: // From 157.5 to 22.5
```

```
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 19: // From 135 to 0
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 20: // From 112.5 to 337.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 21: // From 90 to 315
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterX >= allRoutes.get(i).getXRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 22: // From 67.5 to 292.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 23: // From 45 to 270
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterX >= allRoutes.get(i).getXRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 24: // From 22.5 to 247.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 25: // From 0 to 225
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 26: // From 337.5 to 202.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 27: // From 315 to 180
            for ( int i = 0; i < allRoutes.size(); i++ )
```

```
                    if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                            allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 28: // From 292.5 to 157.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                            allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 29: // From 270 to 135
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterX <= allRoutes.get(i).getXRouteCenter() ||
                            allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 30: // From 247.5 to 112.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                            allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 31: // From 225 to 90
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterX <= allRoutes.get(i).getXRouteCenter() ||
                            allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 32: // From 202.5 to 67.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                            allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
// POLICIES 33 TO 48: COVERING 180
case 33: // From 0 to 180 (North)
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() )
                            fRoutes.add(allRoutes.get(i));
            break;
case 34: // From 337.5 to 157.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 35: // From 315 to 135
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 36: // From 292.5 to 112.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 37: // From 270 to 90 (East)
            for ( int i = 0; i < allRoutes.size(); i++ )
```

```
                    if( vrpCenterX <= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
            break;
case 38: // From 247.5 to 67.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 39: // From 225 to 45
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 40: // From 202.5 to 22.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 41: // From 180 to 0 (South)
            for (int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
            break;
case 42: // From 157.5 to 337.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 43: // From 135 to 315
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 44: // From 112.5 to 292.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 45: // 90 to 270 (West)
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterX >= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
            break;
case 46: // From 67.5 to 247.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 47: // From 45 to 225
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 48: // From 22.5 to 202.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
// POLICIES 49 TO 52: COVERING 2*90
case 49: // From 0 to 90 and 180 to 270 (1st and 3rd quadrants)
```

```
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) ||
                       (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 50: // From 337.5 to 67.5 and 157.5 to 247.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 51: // From 315 to 45 and 135 to 225
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 52: // From 292.5 to 22.5 and 112.5 to 202.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 53: // From 270 to 0 and 90 to 180 (2nd and 4rd quadrants)
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) ||
                       (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 54: // From 247.5 to 337.5 and 67.5 to 157.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 55: // From 225 to 315 and 45 to 135
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
```

```
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 56: // From 202.5 to 292.5 and 22.5 to 112.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 57: // (2nd quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                case 58: // (4rd quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 59: // (1st quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 60: // (3rd quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                default:
                        System.out.println("UNDEFINED POLICY");
                        break;
        }

        return fRoutes;
}


/*********************************************************************************
 * PRIVATE METHOD calcSubSol()
 * Returns the sub-solution containing all given routes
 *********************************************************************************/

private Solution calcSubSol(ArrayList<Route> routes)
{
        Solution sol = new Solution();

        for (int i = 0; i < routes.size(); i++ )
        {
                Route iRoute = routes.get(i);
                sol.addRoute(iRoute);
                sol.addCosts(iRoute);
        }

        return sol;
}
```

```
	/*****************************************************************************
	 * PRIVATE METHOD calcBackRoutes()
	 * Returns those routes from allRoutes that are not frontRoutes
	 *****************************************************************************/

	private ArrayList<Route> calcBackRoutes(ArrayList<Route> frontRoutes,
				ArrayList<Route> allRoutes)
	{
		ArrayList<Route> bRoutes = new ArrayList<Route>();

		for ( int i = 0; i < allRoutes.size(); i++ )
		{
			Route iRoute = allRoutes.get(i);
			if ( frontRoutes.contains(iRoute) == false )
				bRoutes.add(iRoute);
		}

		return bRoutes;
	}


	/*****************************************************************************
	 * PRIVATE METHOD unifySubSols()
	 * Returns the solution that results from the union between two half-solutions
	 *****************************************************************************/

	private Solution unifySubSols(Solution frontSubSol, Solution backSubSol)
	{
		Solution vrpSol = new Solution();

		for ( int i = 0; i < frontSubSol.getRoutes().size(); i++ )
		{
			Route iRoute = frontSubSol.getRoutes().get(i);
			vrpSol.addRoute(iRoute);
			vrpSol.addCosts(iRoute);
		}

		for ( int j = 0; j < backSubSol.getRoutes().size(); j++ )
		{
			Route jRoute = backSubSol.getRoutes().get(j);
			vrpSol.addRoute(jRoute);
			vrpSol.addCosts(jRoute);
		}

		return vrpSol;
	}

}
```

Clase CVRPCalculator

```
package Algoritmo;

import java.util.ArrayList;
import java.util.Random;

import umontreal.iro.lecuyer.randvar.RandomVariateGen;

/*****************************************************************************
 * Project SimORouting - AlgorithmSplit.java
 *
 * This class encapsulates the splitting methodology.
 *****************************************************************************/

public class AlgorithmSplit
{
        /*****************************************************************************
   * INSTANCE FIELDS
   * ***************************************************************************/

        private int firstPolicy = 17; // First splitting policy
        private int lastPolicy = 57; // Last splitting policy
        private ArrayList <Edge> savingsList; // Original savingsList
        private AlgorithmRandCWS splitRCWS; // new instance of the RandCWS algorithm

        private float[] vrpCenter; // (x-bar, y-bar) is a geometric center for the VRP
        private float vrpCenterX; // x-bar = mean of x[i]
        private float vrpCenterY; // y-bar = mean of y[i]

        private ArrayList<Route> frontRoutes; // list of routes satisfying policy
        private ArrayList<Route> backRoutes; // routes not satisfying split policy
        private ArrayList<Node> frontNodes; // nodes in frontRoutes
        private Solution frontSubSol; // sub-solution derived from the frontRoutes
        private Solution backSubSol; // vrpSol = union(frontSubSol, backSubSol)


  /*****************************************************************************
   * CLASS CONSTRUCTOR
   * ***************************************************************************/

        public AlgorithmSplit(ArrayList<Node> vrpNodes, long[][] cMatrix,
                ArrayList<Edge> sList, float vCap, int maxCosts, int sCosts, Random rngJ,
                RandomVariateGen rngL, boolean useL, float bMin, float bMax )
        {
                // Set some instance fields
                savingsList = sList;
                splitRCWS = new AlgorithmRandCWS(cMatrix, vCap, maxCosts, sCosts,
                                rngJ, rngL, useL, bMin, bMax);


                /*****************************************************************************
           * GET THE (X-BAR, Y-BAR) COORDINATES FOR A CENTRAL VRP POINT
           *****************************************************************************/

                vrpCenter = calcGeometricCenter(vrpNodes);
        vrpCenterX = vrpCenter[0]; // x-bar coordinate
                vrpCenterY = vrpCenter[1]; // y-bar coordinate
        }


         /*****************************************************************************
   * PUBLIC METHOD improveSolUsingSplitting()
   *****************************************************************************/

  public Solution improveSolUsingSplitting(Solution vrpSol, int nIterPerSplit,
                RouteCache rc)
  {
```

```
// 1. Initialize dynamic currentVRPSol variable
Solution currentVRPSol = vrpSol;

// 2. For each possible splitting policy, split and solve
for ( int policy = firstPolicy; policy <= lastPolicy; policy++ )
{
        // 2.1. Construct a list with all routes in the currentVRPSol;
        ArrayList<Route> allRoutes = currentVRPSol.getRoutes();

        // 2.2. Set a geometric center for each route in the list
        setRoutesGeometricCenters(allRoutes);

        // 2.3. Split allRoutes in two subsets,
        // (frontRoutes will be the ones we will try to improve next, but
        //  we also need to know which are the complementary backRoutes)
        frontRoutes = calcFrontRoutes(allRoutes, policy);
        backRoutes = calcBackRoutes(frontRoutes, allRoutes);

        // 2.4. Continue this process only if there are enough routes
        if ( frontRoutes.size() > 1 && backRoutes.size() > 0 )
        {
                // 2.5. Construct the initial complementary sub-solutions
                frontSubSol = calcSubSol(frontRoutes);
                backSubSol = calcSubSol(backRoutes);

                // 2.6. Get lists of nodes associated with the selected routes
                frontNodes = frontSubSol.getNodes();

                // 2.7. Calculate splitSavingsList from original savingsList
                //  (notice that the resulting sublist will be already sorted)
                ArrayList<Edge> splitSavingsListF = new ArrayList<Edge>();
                for ( int i = 0; i < savingsList.size(); i++ )
                {
                        Edge iEdge = savingsList.get(i);
                        Node iEdgeOrigin = iEdge.getOrigin();
                        Node iEdgeEnd = iEdge.getEnd();
                        // Construct frontNodes list
                        if ( iEdgeOrigin.getId() < iEdgeEnd.getId() &&
                                        frontNodes.contains(iEdgeOrigin) &&
                                        frontNodes.contains(iEdgeEnd) )
                                splitSavingsListF.add(iEdge);
                }

                // 2.8. Start a random search process to improve frontSubSol
                int nIterF = nIterPerSplit; // * frontRoutes.size();
                for ( int i = 1; i <= nIterF; i++ )
                {
                        // 2.8.1. Get a random solution based on the Random CWS
                        Solution newSubSolF = splitRCWS.constructRandomSol(
                                        frontNodes, splitSavingsListF, true);

                        // 2.8.2. Apply the hash-table improvement and update sol
                        // (this can also help to quickly improve cached values)
                        newSubSolF = rc.improveRoutesUsingHashTable(newSubSolF);
                        newSubSolF = rc.improveAreaUsingHashTable(newSubSolF);

                        // 2.8.3. If appropriate, update frontSubSol
                        if ( newSubSolF.getCosts() < frontSubSol.getCosts() )
                                frontSubSol = newSubSolF;
                }

                // 2.9. Try a quick improvement of backSubSol
                //  (notice that improveRoutesUsingHashTable has been already
                //   applied to this set of routes in the SRGCWSCS.java)
                // backSubSol = rc.improveRoutesUsingHashTable(backSubSol);
                backSubSol = rc.improveAreaUsingHashTable(backSubSol);

                // 2.10. Re-construct the full VRP solution by using frontSubSol
                Solution newSol = unifySubSols(frontSubSol, backSubSol);
```

```
                        // 2.11. If appropriate, update currentVRPSol
                        if ( newSol.getCosts() < currentVRPSol.getCosts() &&
                                newSol.getNodes().size() == currentVRPSol.getNodes().size() &&
                                newSol.getDemand() - currentVRPSol.getDemand() <=
                                        SRGCWSCS.EPSILON )
                                currentVRPSol = newSol;
                }
        }

        // 3. Return the best full VRP solution found so far
        return currentVRPSol;
}


        /*****************************************************************************
         * PRIVATE METHOD setRoutesGeometricCenters()
         * Given a list of routes, it assigns to each route a geometric center
         *****************************************************************************/

        private void setRoutesGeometricCenters(ArrayList<Route> routes)
        {
                for ( int i = 0; i < routes.size(); i++ )
        {
                Route iRoute = routes.get(i);
                ArrayList<Node> iRouteNodes = iRoute.getNodes();
                float[] iRouteCenter = calcGeometricCenter(iRouteNodes);
                iRoute.setXRouteCenter(iRouteCenter[0]);
                iRoute.setYRouteCenter(iRouteCenter[1]);
        }
        }


        /*****************************************************************************
         * PRIVATE METHOD calcGeometricCenter()
         * Returns a geometric center for a set of nodes
         *****************************************************************************/

        private float[] calcGeometricCenter(ArrayList<Node> nodes)
        {
                // 1. Declare and initialize variables
                float sumX = (float) 0.0; // sum of x[i]
                float sumY = (float) 0.0; // sum of y[i]
                float[] center = new float[2]; // center as (x, y) coordinates

                // 2. Calculate sums of x[i] and y[i] for all iNodes in nodes
                Node iNode; // iNode = ( x[i], y[i] )
                for (int i = 0; i < nodes.size(); i++)
                {
                        iNode = nodes.get(i);
                        sumX = sumX + iNode.getX();
                        sumY = sumY + iNode.getY();
                }

                // 3. Calculate means for x[i] and y[i]
                center[0] = sumX / nodes.size(); // mean for x[i]
                center[1] = sumY / nodes.size(); // mean for y[i]

                // 4. Return center as (x-bar, y-bar)
                return center;
        }


        /*****************************************************************************
         * PRIVATE METHOD calcFrontRoutes()
         * Returns those routes from allRoutes that satisfy a given policy
         *****************************************************************************/

         private ArrayList<Route> calcFrontRoutes(ArrayList<Route> allRoutes,
```

```
                    int policy)
{
        ArrayList<Route> fRoutes = new ArrayList<Route>();
        double mPId8 = Math.tan(Math.PI / 8);
        double m3PId8 = Math.tan(3 * Math.PI / 8);

        switch (policy)
        {
                // POLICIES 1 TO 16: COVERING 270
                case 1: // From 180 to 90 (South & East)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                                        vrpCenterX <= allRoutes.get(i).getXRouteCenter() )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 2: // From 157.5 to 67.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 3: // From 135 to 45
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 4: // From 112.5 to 22.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 5: // From 90 to 0 (West & South)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                                        vrpCenterX >= allRoutes.get(i).getXRouteCenter() )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 6: // From 67.5 to 337.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 7: // From 45 to 315
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 8: // From 22.5 to 292.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                                   (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
```

```
case 9: // From 0 to 270 (North & West)
        for (int i = 0; i < allRoutes.size(); i++ )
                if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                        vrpCenterX >= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
        break;
case 10: // From 337.5 to 247.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 11: // From 315 to 225
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 12: // From 292.5 to 202.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 13: // From 270 to 180 (North & East)
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                        vrpCenterX <= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
        break;
case 14: // From 247.5 to 157.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 15: // From 225 to 135
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 16: // From 202.5 to 112.5
        for ( int i = 0; i < allRoutes.size(); i++ )
                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
// POLICIES 17 TO 32: COVERING 225
case 17: // From 180 to 45
        for ( int i = 0; i < allRoutes.size(); i++ )
                if(  vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                         allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                  (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
        break;
case 18: // From 157.5 to 22.5
```

```
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 19: // From 135 to 0
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 20: // From 112.5 to 337.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 21: // From 90 to 315
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterX >= allRoutes.get(i).getXRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 22: // From 67.5 to 292.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 23: // From 45 to 270
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterX >= allRoutes.get(i).getXRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 24: // From 22.5 to 247.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 25: // From 0 to 225
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 26: // From 337.5 to 202.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                    (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                              allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                              (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                              fRoutes.add(allRoutes.get(i));
            break;
case 27: // From 315 to 180
            for ( int i = 0; i < allRoutes.size(); i++ )
```

```
                    if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() ||
                            allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 28: // From 292.5 to 157.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                            allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 29: // From 270 to 135
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterX <= allRoutes.get(i).getXRouteCenter() ||
                            allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 30: // From 247.5 to 112.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                            allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 31: // From 225 to 90
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterX <= allRoutes.get(i).getXRouteCenter() ||
                            allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 32: // From 202.5 to 67.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                            allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                            (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
// POLICIES 33 TO 48: COVERING 180
case 33: // From 0 to 180 (North)
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterY <= allRoutes.get(i).getYRouteCenter() )
                            fRoutes.add(allRoutes.get(i));
            break;
case 34: // From 337.5 to 157.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 35: // From 315 to 135
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 36: // From 292.5 to 112.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                            fRoutes.add(allRoutes.get(i));
            break;
case 37: // From 270 to 90 (East)
            for ( int i = 0; i < allRoutes.size(); i++ )
```

```
                    if( vrpCenterX <= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
            break;
case 38: // From 247.5 to 67.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 39: // From 225 to 45
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 40: // From 202.5 to 22.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 41: // From 180 to 0 (South)
            for (int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterY >= allRoutes.get(i).getYRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
            break;
case 42: // From 157.5 to 337.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 43: // From 135 to 315
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 44: // From 112.5 to 292.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 45: // 90 to 270 (West)
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( vrpCenterX >= allRoutes.get(i).getXRouteCenter() )
                        fRoutes.add(allRoutes.get(i));
            break;
case 46: // From 67.5 to 247.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 47: // From 45 to 225
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
case 48: // From 22.5 to 202.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                    if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                      (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                        fRoutes.add(allRoutes.get(i));
            break;
// POLICIES 49 TO 52: COVERING 2*90
case 49: // From 0 to 90 and 180 to 270 (1st and 3rd quadrants)
```

```
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) ||
                       (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 50: // From 337.5 to 67.5 and 157.5 to 247.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 51: // From 315 to 45 and 135 to 225
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 52: // From 292.5 to 22.5 and 112.5 to 202.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                       allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                       allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 53: // From 270 to 0 and 90 to 180 (2nd and 4rd quadrants)
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) ||
                       (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 54: // From 247.5 to 337.5 and 67.5 to 157.5
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + m3PId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY - mPId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + m3PId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - mPId8 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                       fRoutes.add(allRoutes.get(i));
            break;
case 55: // From 225 to 315 and 45 to 135
            for ( int i = 0; i < allRoutes.size(); i++ )
                  if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + 1 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY - 1 *
                       (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
```

```
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - 1 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 56: // From 202.5 to 292.5 and 22.5 to 112.5
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( allRoutes.get(i).getYRouteCenter() >= vrpCenterY + mPId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                                        allRoutes.get(i).getYRouteCenter() >= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) ||
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY + mPId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) &&
                                        allRoutes.get(i).getYRouteCenter() <= vrpCenterY - m3PId8 *
                                        (allRoutes.get(i).getXRouteCenter() - vrpCenterX) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 57: // (2nd quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                case 58: // (4rd quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 59: // (1st quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY <= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX <= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                case 60: // (3rd quadrant)
                        for ( int i = 0; i < allRoutes.size(); i++ )
                                if( (vrpCenterY >= allRoutes.get(i).getYRouteCenter() &&
                                        vrpCenterX >= allRoutes.get(i).getXRouteCenter()) )
                                        fRoutes.add(allRoutes.get(i));
                        break;
                default:
                        System.out.println("UNDEFINED POLICY");
                        break;
        }

        return fRoutes;
}


/*********************************************************************************
 * PRIVATE METHOD calcSubSol()
 * Returns the sub-solution containing all given routes
 *********************************************************************************/

private Solution calcSubSol(ArrayList<Route> routes)
{
        Solution sol = new Solution();

        for (int i = 0; i < routes.size(); i++ )
        {
                Route iRoute = routes.get(i);
                sol.addRoute(iRoute);
                sol.addCosts(iRoute);
        }

        return sol;
}
```

```
	/*****************************************************************************
	 * PRIVATE METHOD calcBackRoutes()
	 * Returns those routes from allRoutes that are not frontRoutes
	 *****************************************************************************/

	private ArrayList<Route> calcBackRoutes(ArrayList<Route> frontRoutes,
				ArrayList<Route> allRoutes)
	{
		ArrayList<Route> bRoutes = new ArrayList<Route>();

		for ( int i = 0; i < allRoutes.size(); i++ )
		{
			Route iRoute = allRoutes.get(i);
			if ( frontRoutes.contains(iRoute) == false )
				bRoutes.add(iRoute);
		}

		return bRoutes;
	}


	/*****************************************************************************
	 * PRIVATE METHOD unifySubSols()
	 * Returns the solution that results from the union between two half-solutions
	 *****************************************************************************/

	private Solution unifySubSols(Solution frontSubSol, Solution backSubSol)
	{
		Solution vrpSol = new Solution();

		for ( int i = 0; i < frontSubSol.getRoutes().size(); i++ )
		{
			Route iRoute = frontSubSol.getRoutes().get(i);
			vrpSol.addRoute(iRoute);
			vrpSol.addCosts(iRoute);
		}

		for ( int j = 0; j < backSubSol.getRoutes().size(); j++ )
		{
			Route jRoute = backSubSol.getRoutes().get(j);
			vrpSol.addRoute(jRoute);
			vrpSol.addCosts(jRoute);
		}

		return vrpSol;
	}

}
```

Clase Edge

```
package Algoritmo;

import Config.Parametros;
import Interfaces.DrawAble;
import Nucleo.DSSNode;
import java.awt.Graphics;

/*******************************************************************************
 * Project SimORouting - Edge.java
 *
 * This class represents an edge connecting two nodes in the VRP
 *
 *******************************************************************************/
public class Edge implements Comparable<Edge>,DrawAble {

   /*******************************************************************************
    *  INSTANCE FIELDS
    *******************************************************************************/
   private static int nInstances = 0; // number of instances
   private int id; // edge ID
   private Node origin; // origin node
   private Node end; // end node
   private long costs; // edge costs (multiplied by intFactor)
   private long savings; // edge savings (Clarke & Wright)
   private int inRoute = 0; // route containing this edge (0 if no route assigned)

   /*******************************************************************************
    *  CLASS CONSTRUCTOR
    *******************************************************************************/
   public Edge(Node originNode, Node endNode) {
      nInstances++;
      id = nInstances;
      origin = originNode;
      end = endNode;
      costs = 0;
      savings = 0;
   }

   /*******************************************************************************
    * PUBLIC METHOD setCosts()
    *******************************************************************************/
   public void setCosts(long edgeCosts) {
      costs = edgeCosts;
   }

   /*******************************************************************************
    * PUBLIC METHOD setSavings()
    *******************************************************************************/
   public void setSavings(long edgeSavings) {
      savings = edgeSavings;
   }

   /*******************************************************************************
    * PUBLIC METHOD setInRoute()
    *******************************************************************************/
   public void setInRoute(int routeId) {
      inRoute = routeId;
   }

   /*******************************************************************************
    * PUBLIC METHOD getOrigin()
    *******************************************************************************/
   public Node getOrigin() {
      return origin;
   }

   /*******************************************************************************
```

```
     * PUBLIC METHOD getEnd()
     ************************************************************************/
    public Node getEnd() {
        return end;
    }

    /************************************************************************
     * PUBLIC METHOD getId()
     ************************************************************************/
    public int getId() {
        return id;
    }

    /************************************************************************
     * PUBLIC METHOD getCosts()
     ************************************************************************/
    public long getCosts() {
        return costs;
    }

    /************************************************************************
     * PUBLIC METHOD getSavings()
     ************************************************************************/
    public long getSavings() {
        return savings;
    }

    /************************************************************************
     * PUBLIC METHOD getInRoute()
     ************************************************************************/
    public int getInRoute() {
        return inRoute;
    }

    /************************************************************************
     * PUBLIC METHOD toString()
     ************************************************************************/
    @Override
    public String toString() {
        String s = "";
        s = s.concat("\nEdge Id: " + this.getId());
        s = s.concat("\nEdge origin: " + this.getOrigin());
        s = s.concat("\nEdge end: " + this.getEnd());
        s = s.concat("\nEdge costs: " + (this.getCosts() / Parametros.intFactor));
        s = s.concat("\nEdge savings: " + (this.getSavings() / Parametros.intFactor));
        return s;
    }

    /************************************************************************
     * PUBLIC METHOD compareTo()
     ************************************************************************/
    public int compareTo(Edge otherEdge) {
        Edge other = otherEdge;
        long s1 = this.getSavings();
        long s2 = other.getSavings();
        if (s1 < s2) {
            return -1;
        } else if (s1 > s2) {
            return 1;
        }
        return 0;
    }

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel, float Escala, int DesplazamientoX, int DesplazamientoY, int
MaxXEscalada, int MaxYEscalada) {
        DSSNode origen = (DSSNode) this.origin;
        DSSNode fin = (DSSNode) this.end;
        origen.draw(g, AlturaPanel, AnchuraPanel, Escala, DesplazamientoX, DesplazamientoY, MaxXEscalada, MaxYEscalada);
        fin.draw(g, AlturaPanel, AnchuraPanel, Escala, DesplazamientoX, DesplazamientoY, MaxXEscalada, MaxYEscalada);
```

```
            g.drawLine(origen.getXEnPanel(), origen.getYEnPanel(), fin.getXEnPanel(), fin.getYEnPanel());
        }
    }
```

Clase ElapsedTime

```
package Algoritmo;

/*******************************************************************************
 * Project SimORouting - ElapsedTime.java
 *
 * Given an elapsed time in seconds, the method printHMS() of this class returns
 *  an string with the equivalent time in hours, minutes and seconds.
 *
 * Given an elapsed time in seconds, the method doPause() of this class forces
 *  the code to do a pause for a time interval of that size.

 *******************************************************************************/

public class ElapsedTime
{
	public ElapsedTime() {}

	public static String calcHMS(int timeInSeconds)
	{
		String s = "";

		int hours, minutes, seconds;
		hours = timeInSeconds / 3600;
		timeInSeconds = timeInSeconds - (hours * 3600);
		minutes = timeInSeconds / 60;
		timeInSeconds = timeInSeconds - (minutes * 60);
		seconds = timeInSeconds;

		s = s + hours + "h " + minutes + "m " + seconds + "s";

		return s;
	}

	public static void doPause(int timeInSeconds)
	{
		long t0, t1;

		t0 = System.currentTimeMillis();
		t1 = System.currentTimeMillis() + (timeInSeconds * 1000);

		do
		{
			t0 = System.currentTimeMillis();
		} while (t0 < t1);
	}

}
```

Clase Node

```
package Algoritmo;

/*******************************************************************************
 * Project SimORouting - Node.java
 *
 * This class represents a node in the VRP
 *
 *******************************************************************************/
public class Node {

   /*****************************************************************************
    * INSTANCE FIELDS
    *****************************************************************************/
   protected int id; // node ID
   protected int x; // node x coordinate
   protected int y; // node y coordinate
   protected int demand; // node demand
   protected int inRoute; // id of route containing the node
   protected boolean isInterior; // interior -> no direct connection with depot

   /*****************************************************************************
    * CLASS CONSTRUCTOR
    *****************************************************************************/
   public Node(int nodeId, int nodeX, int nodeY, int nodeDemand) {
      id = nodeId;
      x = nodeX;
      y = nodeY;
      demand = nodeDemand;
      inRoute = -1; // node is not assigned to any route yet
      isInterior = false; // initially, all nodes are exterior ones
   }

   /*****************************************************************************
    * PUBLIC METHOD setId()
    *****************************************************************************/
   public void setId(int nodeId) {
      id = nodeId;
   }

   /*****************************************************************************
    * PUBLIC METHOD setX()
    *****************************************************************************/
   public void setX(int nodeX) {
      x = nodeX;
   }

   /*****************************************************************************
    * PUBLIC METHOD setY()
    *****************************************************************************/
   public void setY(int nodeY) {
      y = nodeY;
   }

   /*****************************************************************************
    * PUBLIC METHOD setDemand()
    *****************************************************************************/
   public void setDemand(int nodeDemand) {
      demand = nodeDemand;
   }

   /*****************************************************************************
    * PUBLIC METHOD setInRoute()
    *****************************************************************************/
   public void setInRoute(int routeId) {
      inRoute = routeId;
   }
```

```
    /*****************************************************************************
     * PUBLIC METHOD setIsInterior()
     *****************************************************************************/
    public void setIsInterior(boolean value) {
        isInterior = value;
    }

    /*****************************************************************************
     * PUBLIC METHOD getId()
     *****************************************************************************/
    public int getId() {
        return id;
    }

    /*****************************************************************************
     * PUBLIC METHOD getX()
     *****************************************************************************/
    public int getX() {
        return x;
    }

    /*****************************************************************************
     * PUBLIC METHOD getY()
     *****************************************************************************/
    public int getY() {
        return y;
    }

    /*****************************************************************************
     * PUBLIC METHOD getDemand()
     *****************************************************************************/
    public int getDemand() {
        return demand;
    }

    /*****************************************************************************
     * PUBLIC METHOD getInRoute()
     *****************************************************************************/
    public int getInRoute() {
        return inRoute;
    }

    /*****************************************************************************
     * PUBLIC METHOD isInterior()
     *****************************************************************************/
    public boolean isInterior() {
        return isInterior;
    }

    /*****************************************************************************
     * PUBLIC METHOD toString()
     *****************************************************************************/
    @Override
    public String toString() {
        String s = "";
        s = s.concat(this.getId() + " ");
        s = s.concat(this.getX() + " ");
        s = s.concat(this.getY() + " ");
        s = s.concat(this.getDemand() + "");
        return s;
    }
}
```

Clase Route

```
package Algoritmo;

import Config.Parametros;
import Interfaces.DrawAble;
import java.awt.Graphics;
import java.util.ArrayList;

/*****************************************************************************
 * Project SimORouting - Route.java
 *
 * This class represents a route (array of edges) of the VRP
 *****************************************************************************/
public class Route implements DrawAble {

  /*****************************************************************************
   *  INSTANCE FIELDS
   *****************************************************************************/
  private static int nInstances = 0; // number of instances
  private int id; // route id
  private float costs; // route total costs
  private int demand; // route total demand
  private ArrayList<Edge> edges; // edges list
  ArrayList<Node> nodes; // nodes in route
  private float xRouteCenter; // x-coordinate of the route center
  private float yRouteCenter; // y-coordinate of the route center

  /*****************************************************************************
   *  CLASS CONSTRUCTOR
   *****************************************************************************/
  public Route() {
     nInstances++;
     id = nInstances;
     costs = 0;
     demand = 0;
     edges = new ArrayList<Edge>();
     nodes = new ArrayList<Node>();
     xRouteCenter = (float) 0.0;
     yRouteCenter = (float) 0.0;
  }

  /*****************************************************************************
   *  PUBLIC METHOD addEdge()
   *****************************************************************************/
  public void addEdge(Edge anEdge) {
     edges.add(anEdge);

     if (nodes.contains(anEdge.getOrigin()) == false) {
        nodes.add(anEdge.getOrigin());
     }

     if (nodes.contains(anEdge.getEnd()) == false) {
        nodes.add(anEdge.getEnd());
     }
  }

  /*****************************************************************************
   *  PUBLIC METHOD removeEdge()
   *****************************************************************************/
  public void removeEdge(Edge anEdge) {
     edges.remove(anEdge);
  }

  /*****************************************************************************
   *  PUBLIC METHOD addCosts()
   *****************************************************************************/
  public void addCosts(Edge anEdge) {
     costs += anEdge.getCosts();
```

```
}

/*****************************************************************************
 *  PUBLIC METHOD substractCosts()
 *****************************************************************************/
public void substractCosts(Edge anEdge) {
   costs -= anEdge.getCosts();
}

/*****************************************************************************
 *  PUBLIC METHOD setCosts()
 *****************************************************************************/
public void setCosts(float c) {
   costs = c;
}

/*****************************************************************************
 *  PUBLIC METHOD setXRouteCenter()
 *****************************************************************************/
public void setXRouteCenter(float x) {
   xRouteCenter = x;
}

/*****************************************************************************
 *  PUBLIC METHOD setYRouteCenter()
 *****************************************************************************/
public void setYRouteCenter(float y) {
   yRouteCenter = y;
}

/*****************************************************************************
 *  PUBLIC METHOD addDemand()
 *****************************************************************************/
public void addDemand(Edge anEdge) {
   Node endNode = anEdge.getEnd();
   demand += endNode.getDemand();
}

/*****************************************************************************
 *  PUBLIC METHOD getCosts()
 *****************************************************************************/
public float getCosts() {
   return costs;
}

/*****************************************************************************
 *  PUBLIC METHOD getDemand()
 *****************************************************************************/
public int getDemand() {
   return demand;
}

/*****************************************************************************
 *  PUBLIC METHOD getId()
 *****************************************************************************/
public int getId() {
   return id;
}

/*****************************************************************************
 *  PUBLIC METHOD getEdges()
 *****************************************************************************/
public ArrayList<Edge> getEdges() {
   return edges;
}

/*****************************************************************************
 *  PUBLIC METHOD getNodes()
 *****************************************************************************/
```

```
    public ArrayList<Node> getNodes() {
        ArrayList<Node> n = new ArrayList<Node>();
        for (Edge e : edges) {
            n.add(e.getOrigin());
        }
        return n;
    }

    public String showNodes() {
        String route = "";
        for (Node n : getNodes()) {
            route = route + " " + n.getId();
        }
        return route + "\n";
    }

    /*****************************************************************************
     *  PUBLIC METHOD getXRouteCenter()
     *****************************************************************************/
    public float getXRouteCenter() {
        return xRouteCenter;
    }

    /*****************************************************************************
     *  PUBLIC METHOD getYRouteCenter()
     *****************************************************************************/
    public float getYRouteCenter() {
        return yRouteCenter;
    }

    /*****************************************************************************
     *  PUBLIC METHOD toString()
     *****************************************************************************/
    @Override
    public String toString() {
        String s = "";
        s = s.concat("\nRuta Id: " + this.getId());
        s = s.concat("\nRute costs: " + (this.getCosts() / Parametros.intFactor));
        s = s.concat("\nRuta demand:" + this.getDemand());
        s = s.concat("\nRuta edges: " + this.getEdges());
        s = s + "\n";
        for (Edge e : getEdges()) {
            s = s + " " + e.getOrigin().getId();
        }
        return s + "\n";
    }

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel, float Escala, int DesplazamientoX, int DesplazamientoY, int
MaxXEscalada, int MaxYEscalada) {
        for (Edge e : edges){
            e.draw(g,AlturaPanel,AnchuraPanel, Escala, DesplazamientoX
                    ,DesplazamientoY, MaxXEscalada, MaxYEscalada);
        }
    }
}
```

Clase RouteCache

```
package Algoritmo;

import java.util.Arrays;
import java.util.Hashtable;
import java.util.ArrayList;

/*****************************************************************************
 * Project SimORouting - RouteCache.java
 *
 * Two routes are said to be "nodes-equivalent" if they contain the same set of
 *  nodes (notice that a set of nodes can be sorted in different ways).
 *
 * This class uses a hash table to save the best known-so-far route for each set of
 *  nodes. Therefore, given a route we can try to improve it by looking up at the
 *  hash table, which could contain a better nodes-equivalent route.
 *****************************************************************************/

public class RouteCache
{

	/*****************************************************************************
   * INSTANCE FIELDS
   * ***************************************************************************/

	// Declare and define some basic parameters of the hash table
	private Hashtable<Integer, Route> tableR;
	private Hashtable<Integer, Solution> tableS;
	private int initialCapacity = 1000;
     private float loadFactor = (float) 0.8;
     private int tableSize;
     private long[][] cMatrix;


	/*****************************************************************************
   * CLASS CONSTRUCTOR
   *****************************************************************************/

	public RouteCache(long[][] costsMatrix)
	{
		// Initialize the hash table
		tableR = new Hashtable<Integer, Route>(initialCapacity, loadFactor);
		tableS = new Hashtable<Integer, Solution>(initialCapacity, loadFactor);
		tableSize = 0;
		cMatrix = costsMatrix;
	}


	/*****************************************************************************
   * PUBLIC METHOD improveRoutesUsingHashTable()
   * Given a solution, it tries to enhance it by improving each of its routes.
   * To improve each route it uses the findImprovedRoute() method, which
   *  implements a hash table. Finally, it returns the improved solution.
   *****************************************************************************/

	public Solution improveRoutesUsingHashTable(Solution sol)
	{
		// 1. Reset total costs of the given solution
		sol.setCosts(0);

		// 2. For each route in the given solution, try to improve it
		for ( int i = 0; i < sol.getRoutes().size(); i++ )
	{
		// 2.1. Get a route iRoute in the solution
			Route iRoute = sol.getRoutes().get(i);

			// 2.2. Try to improve iRoute by using the hash table
			//   (if there isn't a better equivalent route in the current hash
```

```
                    //   table, it returns iRoute after adding iRoute to the hash table)
                    Route betterRoute = findImprovedRoute(iRoute);

                    // 2.3. If betterRoute outperforms iRoute, update solution
                if ( betterRoute.getCosts() < iRoute.getCosts() )
                {
                    sol.getRoutes().remove(i);
                    sol.getRoutes().add(i, betterRoute);
                }

                // 2.4. Add costs of the best-found route to the solution
                sol.addCosts(betterRoute);
        }

                // 3. Return the updated solution
                return sol;
        }


        /*****************************************************************************
* PUBLIC METHOD improveAreaUsingHashTable()
* Given a solution, it tries to enhance it by improving each of its routes.
* To improve each route it uses the findImprovedRoute() method, which
*  implements a hash table. Finally, it returns the improved solution.
*****************************************************************************/

        public Solution improveAreaUsingHashTable(Solution sol)
        {
                // 1. Try to improve the area by using the hash table
                Solution betterSol = findImprovedSol(sol);

                // 2. If betterSol outperforms sol, update sol
        if ( betterSol.getCosts() < sol.getCosts() )
                sol = betterSol;

                // 3. Return the updated solution
                return sol;
        }


        /*****************************************************************************
* PUBLIC METHOD getTableRSize()
*****************************************************************************/

        public int getTableRSize()
        {
                tableSize = tableR.size();
                return tableSize;
        }


        /*****************************************************************************
* PUBLIC METHOD getTableSSize()
*****************************************************************************/

        public int getTableSSize()
        {
                tableSize = tableS.size();
                return tableSize;
        }


        /*****************************************************************************
* PRIVATE METHOD findImprovedRoute()
* Given aRoute it returns either an improved route (using the same nodes) from
*  the hash table, or simply aRoute if there isn't a better route in the table
* ***************************************************************************/

        private Route findImprovedRoute(Route aRoute)
```

```
{
        // 1. Obtain the set of nodes contained in aRoute
        int nodes = getNodesFromRoute(aRoute);

        // 2. Obtain, if exists, the cached route containing the same set of nodes
        Route cachedRoute = (Route) tableR.get(nodes);

        // 3. If it does not exist already, save aRoute in cache and return it
        if (cachedRoute == null)
        {
                // before saving aRoute in cache, try to improve the order of its nodes
                aRoute = improveNodesOrder(aRoute);

                tableR.put(nodes, aRoute);
                return aRoute;
        }

        // 4. If it exist already, update the cache if convenient
        else
        {
                // 4.1. If aRoute is better than cachedRoute, save aRoute and return it
                if ( aRoute.getCosts() < cachedRoute.getCosts() )
                {
                        // before saving aRoute in cache, try to improve order of its nodes
                        aRoute = improveNodesOrder(aRoute);

                        tableR.remove(nodes);
                        tableR.put(nodes, aRoute);
                        return aRoute;
                }
                // 4.2. If cachedRoute is better than aRoute, return cachedRoute
                else
                        return cachedRoute;
        }
}


        /*****************************************************************************
* PRIVATE METHOD findImprovedSol()
* Given aSol it returns either an improved sol (using the same nodes) from
*  the hash table, or simply aSol if there isn't a better sol in the table
* ****************************************************************************/

        private Solution findImprovedSol(Solution aSol)
        {
                // 1. Obtain the set of nodes contained in aSol
                int nodes = getNodesFromSol(aSol);

                // 2. Obtain, if exists, the cached sol containing the same set of nodes
                Solution cachedSol = (Solution) tableS.get(nodes);

                // 3. If it does not exist already, save aSol in cache and return it
                if (cachedSol == null)
                {
                        tableS.put(nodes, aSol);
                        return aSol;
                }

                // 4. If it exist already, update the cache if convenient
                else
                {
                        // 4.1. If aSol is better than cachedSol, save aSol and return it
                        if ( aSol.getCosts() < cachedSol.getCosts() )
                        {
                                tableS.remove(nodes);
                                tableS.put(nodes, aSol);
                                return aSol;
                        }
                        // 4.2. If cachedSol is better than aSol, return cachedSol
```

```
                    else
                            return cachedSol;
                }
        }


        /*****************************************************************************
* PRIVATE METHOD getNodesFromRoute()
* Given a route (e.g.: 0-1-2-23-15-0), this method returns a string containing
*  its nodes sorted and separated by semicolons (e.g. "0:0:1:2:15:23"). This
*  string can then be used as a key in a hash table.
* ***************************************************************************/

        private int getNodesFromRoute(Route aRoute)
        {
                // 1. Construct a list edges containing all edges in aRoute
                ArrayList<Edge> edges = aRoute.getEdges();

                // 2. Construct an array to save all nodesID contained in aRoute
                //              (e.g.: route 0-1-2-23-15-0 has 5 edges and 6 nodes)
                int nodesID[] = new int[edges.size() + 1];

                // 3. Select the first edge in the list of edges
                Edge anEdge = edges.get(0);

                // 4. Get the ID of the first node in anEdge
                nodesID[0] = anEdge.getOrigin().getId();

                // 5. Look for every node in aRoute and save its corresponding ID
                int i = 0;
                for( i = 0; i < aRoute.getEdges().size() - 1; i++ )
                {
                        nodesID[ i + 1 ] = anEdge.getEnd().getId();
                        anEdge = edges.get( i + 1 );
                }

                // 6. Close the array by adding node 0 as the last node
                nodesID[ i + 1 ] = anEdge.getEnd().getId();

                // 7. Sort nodes by ID (e.g.: (0,0,1,2,15,23)
                Arrays.sort(nodesID);

                // 8. Construct a string by adding each node in nodesID separated by ":"
                //    (e.g.: ":0:0:1:2:15:23")
                String nodes = "";
                for( i = 0; i < nodesID.length; i++ )
                        nodes = nodes + ":" + nodesID[i];

                // 9. Return the hash code associated with the resulting string
                int hashCode = nodes.hashCode();
                return hashCode;
        }


        /*****************************************************************************
* PRIVATE METHOD getNodesFromSol()
* Given a subsol (e.g.: 0-1-23-0-15-12-0), this method returns a string
*  containing its nodes sorted and separated by semicolons (e.g.
*  "0:0:0:1:12:15:23"). This string can then be used as a key in a hash table.
* ***************************************************************************/

        private int getNodesFromSol(Solution aSol)
        {
                // 1. Construct a list edges containing all edges in aSol
                ArrayList<Edge> edges = new ArrayList<Edge>();
                for ( int i = 0; i < aSol.getRoutes().size(); i++ )
                {
                        Route iRoute = aSol.getRoutes().get(i);
                        edges.addAll(iRoute.getEdges());
```

```
		}

		// 2. Construct an array to save all nodesID contained in aSol
		//		(e.g.: sol 0-1-23-0-15-12-0 has 6 edges and 7 nodes)
		int nodesID[] = new int[edges.size() + 1];

		// 3. Select the first edge in the list of edges
		Edge anEdge = edges.get(0);

		// 4. Get the ID of the first node in anEdge
		nodesID[0] = anEdge.getOrigin().getId();

		// 5. Look for every node in aSol and save its corresponding ID
		int i = 0;
		for( i = 0; i < edges.size() - 1; i++ )
		{
			nodesID[ i + 1 ] = anEdge.getEnd().getId();
			anEdge = edges.get( i + 1 );
		}

		// 6. Close the array by adding node 0 as the last node
		nodesID[ i + 1 ] = anEdge.getEnd().getId();

		// 7. Sort nodes by ID (e.g.: (0,0,0,1,12,15,23)
		Arrays.sort(nodesID);

		// 8. Construct a string by adding each node in nodesID separated by ":"
		//   (e.g.: ":0:0:0:1:12:15:23")
		String nodes = "";
		for( i = 0; i < nodesID.length; i++ )
			nodes = nodes + ":" + nodesID[i];

		// 9. Return the hash code associated with the resulting string
		int hashCode = nodes.hashCode();
		return hashCode;
	}


	/*****************************************************************************
* PRIVATE METHOD improveNodesOrder()
* Given aRoute, this method tries to sort its nodes in a more efficient way.
*  (e.g. by eliminating possible knots in the current route)
* ****************************************************************************/

	private Route improveNodesOrder(Route aRoute)
	{
		ArrayList<Edge> edges = aRoute.getEdges();

		if ( edges.size() >= 4 )
		{
			for ( int i = 0; i <= edges.size() - 3; i++ )
			{
				// Get the current way
				Edge e1 = edges.get(i);
				Edge e2 = edges.get(i + 1);
				Edge e3 = edges.get(i + 2);
				float currentCosts = e1.getCosts() + e2.getCosts() + e3.getCosts();

				// Construct the alternative way
				Node originE1 = e1.getOrigin();
				Node originE2 = e2.getOrigin();
				Node endE2 = e2.getEnd();
				Node endE3 = e3.getEnd();
				Edge e1b = new Edge(originE1, endE2);
				e1b.setCosts(cMatrix[originE1.getId()][endE2.getId()]);
				Edge e2b = new Edge(endE2, originE2);
				e2b.setCosts(cMatrix[endE2.getId()][originE2.getId()]);
				Edge e3b = new Edge(originE2, endE3);
				e3b.setCosts(cMatrix[originE2.getId()][endE3.getId()]);
```

```
                                        float alterCosts = e1b.getCosts() + e2b.getCosts() + e3b.getCosts();

                                        // Compare both ways and, if appropriate, update route
                                        if ( alterCosts < currentCosts )
                                        {
                                                aRoute.removeEdge(e1);
                                                aRoute.substractCosts(e1);
                                                aRoute.getEdges().add(i, e1b);
                                                aRoute.addCosts(e1b);
                                                aRoute.removeEdge(e2);
                                                aRoute.substractCosts(e2);
                                                aRoute.getEdges().add(i + 1, e2b);
                                                aRoute.addCosts(e2b);
                                                aRoute.removeEdge(e3);
                                                aRoute.substractCosts(e3);
                                                aRoute.getEdges().add(i + 2, e3b);
                                                aRoute.addCosts(e3b);
                                        }
                                }
                        }

                        return aRoute;
                }

        }
```

Clase SRGCWSCS

```
package Algoritmo;


import Config.Parametros;
import java.awt.Toolkit;
import java.util.Collections;
import java.util.ArrayList;
import java.util.Random;
import umontreal.iro.lecuyer.probdist.Distribution;
import umontreal.iro.lecuyer.probdist.UniformDist;
import umontreal.iro.lecuyer.randvar.RandomVariateGen;
import umontreal.iro.lecuyer.randvar.UniformGen;
import umontreal.iro.lecuyer.rng.GenF2w32;
import umontreal.iro.lecuyer.rng.RandomStreamBase;

/*****************************************************************************
 * Project SimORouting - SRGCWSCS.java
 *
 * This class contains the main() method, which implements the SR-GCWS-CS
 *  algorithm for the CVRP.
 *****************************************************************************/
public class SRGCWSCS {

   /*************************************************************************
    * STATIC VARIABLES
    *************************************************************************/
   static final double NANOUNITS = 1.0e+9; // For time-unit conversions
   static final int INFINITE = Integer.MAX_VALUE;
   static final float EPSILON = (float) 0.001;
   static String line = "\n*************************************************\n";

   public static ArrayList<Solution> getSolutions(ArrayList<Node> vrpNodes,long[][] costsMatrix) {


      /*************************************************************************
       *  3. CALCULATE COSTS & SAVINGS MATRICES
       *************************************************************************/
      // Costs are calculated as Eclidean distances in the plane. Then, they are
      //  multiplied by intFactor to avoid unnecessary work with float numbers.
      // Savings are computed using the definition from the classical CWS
      //  heuristic.
      CVRPCalculator calculator = new CVRPCalculator(vrpNodes);
      long[][] savingsMatrix = calculator.calcSavingsMatrix(costsMatrix);

      /*************************************************************************
       * 4. CREATE AND SORT THE EFFICIENCY (SAVINGS) LIST
       * While generating the savingsList, each ijEdge in that list is created
       *  and its associated costs and savings are set.
       *************************************************************************/
      ArrayList<Edge> savingsList = new ArrayList<Edge>();

      // 4.1. Create savingsList = { ijEdge / 0 < i < j < dim }
      for (int i = 1; i < vrpNodes.size() - 1; i++) // node 0 is the depot
      {
         for (int j = i + 1; j < vrpNodes.size(); j++) {
            Node iNode = vrpNodes.get(i);
            Node jNode = vrpNodes.get(j);
            Edge ijEdge = new Edge(iNode, jNode);
            ijEdge.setCosts(costsMatrix[i][j]);
            ijEdge.setSavings(savingsMatrix[i][j]);
            savingsList.add(ijEdge);
         }
      }

      // 4.2. Sort edges in savings list from lower to higher savings
      Collections.sort(savingsList);
```

```
/*************************************************************************
 *  5. INITIALIZE THE RANDOM NUMBER GENERATORS FOR THIS PROGRAM
 *************************************************************************/
// 5.1. Initialize a generator based on java.util.Random
Random rngJava = new Random(); // Java random number generator
if (Parametros.Semilla != 0) // if seed = 0 no seed is assigned
{
   rngJava.setSeed(Parametros.Semilla);
}

// 5.2. Initialize a generator based on Lecuyer's SSJ library
RandomStreamBase stream = new GenF2w32(); // L'Ecuyer stream
Distribution dist = new UniformDist(0.0, 1.0);
RandomVariateGen rngLecuyer = new UniformGen(stream, (UniformDist) dist);


/*************************************************************************
 *  6. CREATE AN INSTANCE OF THE RANDOMIZED CWS ALGORITHM
 *************************************************************************/
// This instance will be used both for constructing the CWS solution
//  (useRandomSelection = false) and for constructing each Randomized CWS
//  solution (useRandomSelection = true)
AlgorithmRandCWS rcwsAlg = new AlgorithmRandCWS(costsMatrix, Parametros.CapacidadDelVehiculo,
      Parametros.getLongitudMaximaDeLaRuta(), Parametros.getCosteDelServicio(), rngJava, rngLecuyer,
Parametros.UseLecuyer,
      Parametros.BetaMin, Parametros.BetaMax);


/*************************************************************************
 *  7. CALCULATE THE CLARKE & WRIGHT SOLUTION (PARALLEL VERSION)
 *************************************************************************/
// useRandomSelection = false -> select the edge with the highest savings
boolean useRandomSelection = false;
Solution cwsSol = rcwsAlg.constructRandomSol(vrpNodes, savingsList,
      useRandomSelection);


String s="THE CLARKE & WRIGHT SOLUTION\r\n";
for (int i = 1; i <= cwsSol.getRoutes().size(); i++) {
   Route aRoute = cwsSol.getRoutes().get(i - 1);
   s = s.concat("Route " + i + " || ");
   s = s.concat("Costs = " + (double) aRoute.getCosts() / 10000);
   s = s.concat("\r\n");
}
s = s.concat("\r\n");
for (int i = 1; i <= cwsSol.getRoutes().size(); i++) {
   Route aRoute = cwsSol.getRoutes().get(i - 1);
   for (Edge e : aRoute.getEdges()) {
      if (e.getOrigin().getId() == 0) {
         s = s.concat("0" + " " + e.getEnd().getId());
      } else {
         s = s.concat(" " + e.getEnd().getId());
      }

   }
   s = s + "\r\n";
}
s = s.concat("\r\n\r\n");
System.out.println(s);


float cwsSolCosts = cwsSol.getCosts();
System.out.println(line + "\nWELCOME TO THIS PROGRAM!");
System.out.println("\nCOSTS OF THE CLARKE&WRIGTH SOLUTION: "
      + (cwsSolCosts / Parametros.intFactor));


/*************************************************************************
 *  8. PERFORM ITERATIVE RANDOM SEARCH WITH TWO IMPROVEMENT PROCESSES
```

```
     ************************************************************************/
    // 8.1. DISPLAY A MESSAGE ON THE CONSOLE AND START THE PROCESS CLOCK
    System.out.println(line + "\nRANDOM SEARCH IN PROGRESS, PLEASE WAIT...");
    long startTime = System.nanoTime(); // start time for computations

    // 8.2. SET UP ALL NECESSARY VARIABLES
    // useRandomSelection = true -> select next edge using random criteria
    useRandomSelection = true;
    // vrpSol will contain the current VRP solution
    Solution vrpSol = new Solution();
    // bestSols will contain a list with the best solutions found so far
    ArrayList<Solution> bestSols = new ArrayList<Solution>();
    bestSols.add(cwsSol); // it is not an empty list anymore
    // rCache will contain a hash table with the best routes found so far
    RouteCache rCache = new RouteCache(costsMatrix);
    // splitAlg is an instance of the splitting algorithm
    AlgorithmSplit splitAlg = new AlgorithmSplit(vrpNodes, costsMatrix,
        savingsList, Parametros.CapacidadDelVehiculo, Parametros.getLongitudMaximaDeLaRuta(),
Parametros.getCosteDelServicio(), rngJava,
        rngLecuyer, Parametros.UseLecuyer, Parametros.BetaMin, Parametros.BetaMax);
    // Number of 'promising' solutions (some of them will be repeated)
    int nPromisingSols = 0;

    // 8.3. START ITERATIVE RANDOM-SEARCH-WITH-IMPROVEMENT PROCESS
    for (int i = 1; i <= Parametros.IteracionesExteriores; i++) {
      // 8.3.1. Get a random solution based on the Random CWS algorithm
      vrpSol = rcwsAlg.constructRandomSol(vrpNodes, savingsList,
          useRandomSelection);

      // 8.3.2. Try to quickly improve the solution by using previous results
      //   about best-found-so-far "equivalent" routes saved in a hash table
      vrpSol = rCache.improveRoutesUsingHashTable(vrpSol);

      // 8.3.3. If current solution is a "promising" solution, try to
      //   improve it further by using splitting and, after that,
      //   update the bestSols list
      if (vrpSol.getCosts() < cwsSol.getCosts()) {
        // Update number of promising solutions found
        nPromisingSols++;

        // Set nIterPerSplit = 0 to avoid Split
        if (Parametros.IteracionesSplit > 0) {
          vrpSol = splitAlg.improveSolUsingSplitting(vrpSol,
              Parametros.IteracionesSplit, rCache);
        }
        bestSols = updateBestSolsList(vrpSol, bestSols, Parametros.SolucionesGuardadas,
            startTime, i, nPromisingSols, Parametros.IteracionesSplit, rCache);
      }
    }

    // 8.4. STOP THE PROCESS CLOCK AND DISPLAY A MESSAGE ON THE CONSOLE
    long endTime = System.nanoTime(); // end time for computations
    double elapsed = (endTime - startTime) / NANOUNITS;
    int elapsedInt = (int) Math.round(elapsed);
    System.out.println(line + "\nEND OF RANDOM SEARCH.");
    System.out.println("Tiempo transcurrido: "+ElapsedTime.calcHMS(elapsedInt));



    return bestSols;
  }


  /**************************************************************************
   * STATIC METHOD updateBestSolsList()
   * Given a sol, this method considers it in order to update and sort the
   *  bestSolList. If sol is a new record, it prints a message on the console.
   **************************************************************************/
```

```
   static ArrayList<Solution> updateBestSolsList(Solution sol, ArrayList<Solution> bestSols, int nSols, double startTime, int i, int nPromisingSols,
         int nIterPerSplit, RouteCache rCache) {
      // Compare sol with solutions in the sorted list to determine if sol has
      //   to be added to the list and, if so, in which position
      for (int j = 0; j < bestSols.size(); j++) {
         // If sol outperforms jSolution
         if (sol.getCosts() < bestSols.get(j).getCosts()
               && (j == 0 || sol.getCosts() > bestSols.get(j - 1).getCosts())) {
            // Add sol at the j-position of the list
            bestSols.add(j, sol);
            // If the size limit has been exceeded, delete the last element
            if (bestSols.size() > nSols) {
               bestSols.remove(j + 1);
            }
            // If sol is a new record, send a message to the console
            if (j == 0) {
               //System.out.println("\n\n" + sol.toString());
               float costs = ((float) sol.getCosts()) / Parametros.intFactor;
               System.out.println("New record: " + costs + ". SolID: " + sol.getId() + " (see sol above).");
               double endTime = System.nanoTime();
               double elapsed = (endTime - startTime) / NANOUNITS;
               int elapsedInt = (int) Math.round(elapsed);
               System.out.println("Main-process iterations: " + i);
               System.out.println("Iterations per split: " + nIterPerSplit);
               System.out.println("Promising sols: " + nPromisingSols
                     + " (" + 100 * nPromisingSols / i + "%)");
               System.out.println("CacheR size: " + rCache.getTableRSize());
               System.out.println("CacheS size: " + rCache.getTableSSize());
               System.out.println("Elapsed time: "
                     + ElapsedTime.calcHMS(elapsedInt) + "\nStill working...");
               Toolkit.getDefaultToolkit().beep();
            }
            break; // Done, leave the for bucle
         }
      }

      return bestSols; }}
```

Clase Solution

```
package Algoritmo;

import Interfaces.DrawAble;
import Nucleo.DSSNode;
import java.awt.Graphics;
import java.util.ArrayList;

/*****************************************************************************
 * Project SimORouting - Solution.java
 *
 * This class represents a solution (array of routes) of the VRP
 *****************************************************************************/
public class Solution implements Comparable<Solution>, DrawAble {

   /*************************************************************************
    * INSTANCE FIELDS
    *************************************************************************/
   private static int nInstances = 0; // number of instances
   private int id; // solution ID
   private float costs; // solution costs
   private ArrayList<Route> routes; // list of routes in this solution
   private ArrayList<Node> nodos = new ArrayList<Node>();

   /*************************************************************************
    * CLASS CONSTRUCTOR
    *************************************************************************/
   public Solution() {
      nInstances++;
      id = nInstances;
```

```
    costs = 0;
    routes = new ArrayList<Route>();
}

/*****************************************************************************
 *  PUBLIC METHOD addRoute()
 *****************************************************************************/
public void addRoute(Route aRoute) {
    routes.add(aRoute);
}

/*****************************************************************************
 *  PUBLIC METHOD addCosts()
 *****************************************************************************/
public void addCosts(Route aRoute) {
    costs += aRoute.getCosts();
}

/*****************************************************************************
 *  PUBLIC METHOD setCosts()
 *****************************************************************************/
public void setCosts(float c) {
    costs = c;
}

/*****************************************************************************
 *  PUBLIC METHOD getId()
 *****************************************************************************/
public int getId() {
    return id;
}

/*****************************************************************************
 *  PUBLIC METHOD getCosts()
 *****************************************************************************/
public float getCosts() {
    return costs;
}

/*****************************************************************************
 *  PUBLIC METHOD getRoutes()
 *****************************************************************************/
public ArrayList<Route> getRoutes() {
    return routes;
}

/*****************************************************************************
 *  PUBLIC METHOD getDemand()
 *****************************************************************************/
public float getDemand() {
    float demand = 0;
    for (Route aRoute : this.routes) {
        demand = demand + aRoute.getDemand();
    }

    return demand;
}

public void setNodos(ArrayList<Node> nodos) {
    this.nodos = nodos;
}

/*****************************************************************************
 *  PUBLIC METHOD getNodes()
 *****************************************************************************/
public ArrayList<Node> getNodes() {
    if (this.routes.isEmpty()) {
        return nodos;
    } else {
```

```
        ArrayList<Node> nodes = new ArrayList<Node>();
        ArrayList<Node> candidates = new ArrayList<Node>();

        // In any list of nodes, the first one is the depot
        for (int i = 0; i < routes.size(); i++) {
           Route iRoute = routes.get(i);
           candidates = iRoute.getNodes();
           for (int j = 0; j < candidates.size(); j++) {
              Node jCandidate = candidates.get(j);
              if (nodes.contains(jCandidate) == false) {
                 if (jCandidate.getId() == 0) // the depot must be the first
                 {
                    nodes.add(0, jCandidate);
                 } else {
                    nodes.add(jCandidate);
                 }
              }
           }
        }

        return nodes;
     }
  }



  /*****************************************************************************
   *  PUBLIC METHOD toString()
   *****************************************************************************/
  @Override
  public String toString() {
     Route aRoute; // auxiliary Route variable
     String s = "";
//      s = s.concat("\r\n");
//      s = s.concat("Sol ID : " + this.getId() + "\r\n");
//      s = s.concat("Sol costs: " + (this.getCosts() / SRGCWSCS.intFactor) + "\r\n");
//      s = s.concat("# of routes in sol: " + routes.size());
//      s = s.concat("\r\n\r\n\r\n");
//      s = s.concat("List of routes (cost and nodes): \r\n\r\n");
//      for (int i = 1; i <= routes.size(); i++) {
//         aRoute = routes.get(i - 1);
//         s = s.concat("Route " + i + " || ");
//         s = s.concat("Costs = " + (double) aRoute.getCosts()/SRGCWSCS.intFactor);
//         s = s.concat("\r\n");
//      }
//      s = s.concat("\r\n");
     for (int i = 1; i <= routes.size(); i++) {
        aRoute = routes.get(i - 1);
        for (Edge e : aRoute.getEdges()) {
           if (e.getOrigin().getId() == 0) {
              s = s.concat("0" + " " + e.getEnd().getId());
           } else {
              s = s.concat(" " + e.getEnd().getId());
           }

        }
        s = s + "\r\n";
     }
//      s = s.concat("\r\n\r\n");
     return s;
  }

  /*****************************************************************************
   *  PUBLIC METHOD compareTo()
   *****************************************************************************/
  public int compareTo(Solution otherSol) {
     Solution other = otherSol;
     float e1 = this.getCosts();
     float e2 = other.getCosts();
```

```
        if (e1 < e2) {
            return -1;
        } else if (e1 > e2) {
            return 1;
        }
        return 0;
    }

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel, float Escala, int DesplazamientoX, int DesplazamientoY, int
MaxXEscalada, int MaxYEscalada) {
        if (routes.isEmpty()) {
            for (Node n : nodos) {
                DSSNode nodo = (DSSNode) n;
                nodo.draw(g, AlturaPanel, AnchuraPanel, Escala, DesplazamientoX,
                        DesplazamientoY, MaxXEscalada, MaxYEscalada);
            }
        } else {
            for (Route r : routes) {
                r.draw(g, AlturaPanel, AnchuraPanel, Escala, DesplazamientoX,
                        DesplazamientoY, MaxXEscalada, MaxYEscalada);
            }
        }
    }
}
```

Paquete "Config"

Clase Parametros

```
package Config;

/**
 *
 * @author Administrador
 */
public class Parametros {

    public static int IteracionesExteriores = 200;
    public static int IteracionesSplit = 10;
    public static float BetaMin = (float) 0.1;
    public static float BetaMax = (float) 0.3;
    public static int SolucionesGuardadas = 10;
    public static int Semilla = 0;
    public static boolean UseLecuyer = false;

    public static final float[] ModificadorZona = {(float) 0.05,(float)0.5};
    public static final float[] ModificadorPoblacion = {(float) 1,(float)1.5,(float)2};

    public static int intFactor = 10000;
    public static int CapacidadDelVehiculo;
    private static int LongitudMaximaDeLaRuta;
    private static int CosteDelServicio;

    //Estos valores son los indicados por el usuario en la pantalla principal
    //Modifican el "peso" de las poblaciones y de las zonas naturales
    public static Integer AjustePoblaciones;
    public static Integer AjusteReservas;

    public static void setDefaultValues() {
        IteracionesExteriores = 200;
        IteracionesSplit = 10;
        BetaMin = (float) 0.1;
        BetaMax = (float) 0.3;
        SolucionesGuardadas = 10;
        Semilla = 0;
        UseLecuyer = false;
    }

    public static int getLongitudMaximaDeLaRuta() {
        return LongitudMaximaDeLaRuta;
    }

    public static void setLongitudMaximaDeLaRuta(int LongitudMaximaDeLaRuta) {
        Parametros.LongitudMaximaDeLaRuta = LongitudMaximaDeLaRuta;
        if(Parametros.LongitudMaximaDeLaRuta <= 0){
            Parametros.LongitudMaximaDeLaRuta = Integer.MAX_VALUE;
        }
    }

    public static int getCosteDelServicio() {
        return CosteDelServicio;
    }

    public static void setCosteDelServicio(int CosteDelServicio) {
        Parametros.CosteDelServicio = CosteDelServicio;
    }

}
```

Paquete “DSSInOut”

Clase DSSInOut

```
package DSSInOut;

import Nucleo.DSSNode;
import Nucleo.BasePoblacionDePaso;
import Nucleo.BaseZonaNatural;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

/**
 *
 * @author Administrador
 */
public class DSSInOut {

   // Lee y guarda los Nodos del fichero de nodos
   public static ArrayList<DSSNode> getNodesList(String inputNodesFileName) {
      ArrayList<DSSNode> nodesList = new ArrayList<DSSNode>();
      try {
         BufferedReader reader = new BufferedReader(new FileReader(inputNodesFileName));
         String line;
         while ((line = reader.readLine()) != null) {
            String[] values = mySplit(line);

            int Id = Integer.valueOf(values[0]);
            String Nombre = values[1];
            int X = Integer.valueOf(values[2]);
            int Y = Integer.valueOf(values[3]);
            int Demanda = Integer.valueOf(values[4]);
            int Poblacion = Integer.valueOf(values[5]);

            DSSNode aNode = new DSSNode(Id, Nombre, X, Y, Demanda, Poblacion);
            nodesList.add(aNode);
         }
      } catch (IOException exception) {
         System.out.println("Error processing input nodes file: " + exception);
      }
      return nodesList;
   }

   // Lee y guarda las poblaciones de paso del fichero de poblaciones de paso
   public static ArrayList<BasePoblacionDePaso> getPoblacionesList(String inputPoblacionFileName) {
      ArrayList<BasePoblacionDePaso> poblacionesList = new ArrayList<BasePoblacionDePaso>();
      try {
         BufferedReader reader = new BufferedReader(new FileReader(inputPoblacionFileName));
         String line;
         while ((line = reader.readLine()) != null) {
            String[] values = mySplit(line);

            int IdOrigen = Integer.valueOf(values[0]);
            int IdDestino = Integer.valueOf(values[1]);
            String NombreOrigen = values[2];
            String NombreDestino = values[3];
            String Nombre = values[4];
            int Poblacion = Integer.valueOf(values[5]);
            int Tipo = Integer.valueOf(values[6]);
            int X = Integer.valueOf(values[7]);
            int Y = Integer.valueOf(values[8]);
```

```
            BasePoblacionDePaso aPoblacionDePaso = new BasePoblacionDePaso(IdOrigen, IdDestino, NombreOrigen,
NombreDestino, Nombre, Poblacion, Tipo, X, Y);
            poblacionesList.add(aPoblacionDePaso);
          }
        } catch (IOException exception) {
          System.out.println("Error processing input nodes file: " + exception);
        }
        return poblacionesList;
      }

      // Lee y guarda las zonas naturales del fichero de zonas naturales
      public static ArrayList<BaseZonaNatural> getZonaNaturalList(String inputNodesFileName) {
        ArrayList<BaseZonaNatural> zonaNaturalList = new ArrayList<BaseZonaNatural>();
        try {
          BufferedReader reader = new BufferedReader(new FileReader(inputNodesFileName));
          String line;
          while ((line = reader.readLine()) != null) {
            String[] values = mySplit(line);
            int IdOrigen = Integer.valueOf(values[0]);
            int IdDestino = Integer.valueOf(values[1]);
            String NombreOrigen = values[2];
            String NombreDestino = values[3];
            int Km = Integer.valueOf(values[4]);
            int Tipo = Integer.valueOf(values[5]);
            int X = Integer.valueOf(values[6]);
            int Y = Integer.valueOf(values[7]);

            BaseZonaNatural aZonaNatural = new BaseZonaNatural(IdOrigen, IdDestino, NombreOrigen, NombreDestino, Km, Tipo, X,
Y);
            zonaNaturalList.add(aZonaNatural);
          }
        } catch (IOException exception) {
          System.out.println("Error processing input nodes file: " + exception);
        }
        return zonaNaturalList;
      }

      public static void saveToFile(String text, String file) {
        try {
          File f = new File(file);
          f.createNewFile();
          FileWriter fstream = new FileWriter(f);
          BufferedWriter out = new BufferedWriter(fstream);
          out.write(text);
          out.close();
        } catch (IOException exception) {
          System.out.println("Error processing output file: " + exception);
        }
      }

      // Funcion que trocea la linea e ignora tabuladores de más.
      private static String[] mySplit(String line) {
        String[] OldLine = line.split("\t");
        String[] myLine = new String[OldLine.length];
        int i = 0;
        for (String token : OldLine) {
          if (!token.equals("")) {
            myLine[i] = token;
            i++;
          }
        }
        return myLine;
      }
    }
```

Paquete “Interfaces”

Clase Drawable

```
package Interfaces;

import java.awt.Graphics;

public interface DrawAble {

   public void draw(Graphics g, int AlturaPanel, int AnchuraPanel,
         float Escala, int DesplazamientoX, int DesplazamientoY,
         int MaxXEscalada,int MaxYEscalada);

}
```

Paquete Nucleo

Clase BasePoblacionDePaso

```
package Nucleo;

import Interfaces.DrawAble;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;

/**
 *
 * @author koldo
 */
public class BasePoblacionDePaso implements DrawAble {

    private int IdOrigen;
    private int IdDestino;
    private String NombreOrigen;
    private String NombreDestino;
    private String Nombre;
    private int Poblacion;
    private int Tipo;
    private int X;
    private int Y;
    private int XEnPanel;
    private int YEnPanel;

    public BasePoblacionDePaso(int IdOrigen, int IdDestino, String NombreOrigen, String NombreDestino, String Nombre, int
Poblacion, int Tipo, int X, int Y) {
        this.IdOrigen = IdOrigen;
        this.IdDestino = IdDestino;
        this.NombreOrigen = NombreOrigen;
        this.NombreDestino = NombreDestino;
        this.Nombre = Nombre;
        this.Poblacion = Poblacion;
        this.Tipo = Tipo;
        this.X = X;
        this.Y = Y;
    }

    public int getIdDestino() {
        return IdDestino;
    }

    public int getIdOrigen() {
        return IdOrigen;
    }

    public String getNombre() {
        return Nombre;
    }

    public String getNombreDestino() {
        return NombreDestino;
    }

    public String getNombreOrigen() {
        return NombreOrigen;
    }

    public int getPoblacion() {
        return Poblacion;
```

```
    }

    public int getTipo() {
        return Tipo;
    }

    public int getX() {
        return X;
    }

    public int getY() {
        return Y;
    }

    private void setCoordenadasEnPanel(int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
            int MaxX, int MaxY) {

        float MaxXEscalada = ((float) MaxX + DesplazamientoX) * Escala;
        float MaxYEscalada = ((float) MaxY + DesplazamientoY) * Escala;
        //Calculo el valor del X actual tras desplazarla
        float XDesplazada = this.X + DesplazamientoX;
        //Calculo el valor del X actual tras desplazarla y escalarla
        float XEscalada = XDesplazada * Escala;
        //Calculo el valor del X actual tras desplazarla, escalarla y centrar
        //la imagen. Este sera su valor final
        float XCentrada = XEscalada + (AnchuraPanel - MaxXEscalada) / 2;

        //Calculo el valor del Y actual tras desplazarla
        float YDesplazada = this.Y + DesplazamientoY;
        //Calculo el valor del Y actual tras desplazarla y escalarla
        float YEscalada = YDesplazada * Escala;
        //Calculo el valor del X actual tras desplazarla, escalarla y centrar
        //la imagen.
        float YCentrada = YEscalada + (AlturaPanel - MaxYEscalada) / 2;
        //Finalmente puesto que en el Jpanel el punto 0,0 corresponde a la
        //Esquina superior izquierda tengo que invertir las Y haciendo que su
        //valor final sea el del alto del panel menos el valor calculado
        //puesto que queremos que el margen se respete tanto por arriba como
        //por abajo, para pintarlo, le restamos a la altura del panel la mitad
        //del margen
//      YCentrada = AlturaPanel - this.Margen / 2 - YCentrada;
        YCentrada = AlturaPanel - YCentrada;

        this.XEnPanel = Math.round(XCentrada);
        this.YEnPanel = Math.round(YCentrada);

    }

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
            int MaxXEscalada, int MaxYEscalada) {
        BufferedImage image = null;
        try {
            String file = System.getProperty("user.dir") + "\\src\\dss\\GUI\\Imagenes\\House2.png";
            image = ImageIO.read(new File(file));
        } catch (IOException ex) {
            Logger.getLogger(DSSNode.class.getName()).log(Level.SEVERE, null, ex);
        }
        setCoordenadasEnPanel(AlturaPanel, AnchuraPanel, Escala, DesplazamientoX, DesplazamientoY, MaxXEscalada,
MaxYEscalada);

        g.drawImage(image, XEnPanel, YEnPanel, null);
//      g.fillOval(X,Y, 4, 4);
//      int DesplazamientoX = this.Nombre.length()/2+2;
        //  g.drawString(this.Nombre,X+20,Y+13);


    }}
```

Clase BaseZonaNatural

```
package Nucleo;

import Interfaces.DrawAble;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;

/**
 *
 * @author koldo
 */
public class BaseZonaNatural implements DrawAble {

    private int IdOrigen;
    private int IdDestino;
    private String NombreZonaNaturalOrigen;
    private String NombreZonaNaturalDestino;
    private int KmZonaNatural;
    private int TipoZonaNatural;
    private int X;
    private int Y;
    private int XEnPanel;
    private int YEnPanel;

    public BaseZonaNatural(int IdOrigen, int IdDestino, String NombreZonaNaturalOrigen, String NombreZonaNaturalDestino, int
KmZonaNatural, int TipoZonaNatural, int X, int Y) {
        this.IdOrigen = IdOrigen;
        this.IdDestino = IdDestino;
        this.NombreZonaNaturalOrigen = NombreZonaNaturalOrigen;
        this.NombreZonaNaturalDestino = NombreZonaNaturalDestino;
        this.KmZonaNatural = KmZonaNatural;
        this.TipoZonaNatural = TipoZonaNatural;
        this.X = X;
        this.Y = Y;
    }

    public int getIdDestino() {
        return IdDestino;
    }

    public int getIdOrigen() {
        return IdOrigen;
    }

    public int getKmZonaNatural() {
        return KmZonaNatural;
    }

    public String getNombreZonaNaturalDestino() {
        return NombreZonaNaturalDestino;
    }

    public String getNombreZonaNaturalOrigen() {
        return NombreZonaNaturalOrigen;
    }

    public int getTipoZonaNatural() {
        return TipoZonaNatural;
    }

    public int getX() {
```

```
        return X;
    }

    public int getY() {
        return Y;
    }

    private void setCoordenadasEnPanel(int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
            int MaxX, int MaxY) {

        float MaxXEscalada = ((float) MaxX + DesplazamientoX) * Escala;
        float MaxYEscalada = ((float) MaxY + DesplazamientoY) * Escala;
        //Calculo el valor del X actual tras desplazarla
        float XDesplazada = this.X + DesplazamientoX;
        //Calculo el valor del X actual tras desplazarla y escalarla
        float XEscalada = XDesplazada * Escala;
        //Calculo el valor del X actual tras desplazarla, escalarla y centrar
        //la imagen. Este sera su valor final
        float XCentrada = XEscalada + (AnchuraPanel - MaxXEscalada) / 2;

        //Calculo el valor del Y actual tras desplazarla
        float YDesplazada = this.Y + DesplazamientoY;
        //Calculo el valor del Y actual tras desplazarla y escalarla
        float YEscalada = YDesplazada * Escala;
        //Calculo el valor del X actual tras desplazarla, escalarla y centrar
        //la imagen.
        float YCentrada = YEscalada + (AlturaPanel - MaxYEscalada) / 2;
        //Finalmente puesto que en el Jpanel el punto 0,0 corresponde a la
        //Esquina superior izquierda tengo que invertir las Y haciendo que su
        //valor final sea el del alto del panel menos el valor calculado
        //puesto que queremos que el margen se respete tanto por arriba como
        //por abajo, para pintarlo, le restamos a la altura del panel la mitad
        //del margen
//      YCentrada = AlturaPanel - this.Margen / 2 - YCentrada;
        YCentrada = AlturaPanel - YCentrada;

        this.XEnPanel = Math.round(XCentrada);
        this.YEnPanel = Math.round(YCentrada);

    }

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
            int MaxXEscalada, int MaxYEscalada) {
        BufferedImage image = null;
        try {
            String file = System.getProperty("user.dir") + "\\src\\dss\\GUI\\Imagenes\\arbol1.gif";
            image = ImageIO.read(new File(file));
        } catch (IOException ex) {
            Logger.getLogger(DSSNode.class.getName()).log(Level.SEVERE, null, ex);
        }
        setCoordenadasEnPanel(AlturaPanel, AnchuraPanel, Escala,
                DesplazamientoX, DesplazamientoY, MaxXEscalada, MaxYEscalada);

        g.drawImage(image, XEnPanel, YEnPanel, null);
    }
}
```

Clase DSSCalculator

```
package Nucleo;

import Algoritmo.Edge;
import Algoritmo.Route;
import Algoritmo.Solution;
import Config.Parametros;
import java.util.ArrayList;

/**
 *
 * @author Administrador
 */
public class DSSCalculator {

   int dim;
   long[][] distanciasMatrix;
   long[][] zonasMatrix;
   long[][] poblacionesMatrix;
   long[][] HabPoblacion;
   long[][] KMReserva;
   String newline = System.getProperty("line.separator");

   public DSSCalculator(ArrayList<DSSNode> nodesList) {
      dim = nodesList.size();
      distanciasMatrix = getMatrixOfDistances(nodesList);
      zonasMatrix = new long[dim][dim];
      poblacionesMatrix = new long[dim][dim];

      HabPoblacion = new long[dim][dim];
      KMReserva = new long[dim][dim];
   }

   public long[][] getPoblacionesMatrix() {
      return poblacionesMatrix;
   }

   public void setPoblacionesMatrix(ArrayList<BasePoblacionDePaso> poblaciones) {
      for (BasePoblacionDePaso poblacion : poblaciones) {
         int IdOrigen = poblacion.getIdOrigen();
         int IdDestino = poblacion.getIdDestino();
         int tipo = poblacion.getTipo() - 1;
         int pob = poblacion.getPoblacion();
         this.poblacionesMatrix[IdOrigen][IdDestino] = (long) (pob/400 * Parametros.ModificadorPoblacion[tipo] *
Parametros.intFactor);
         this.poblacionesMatrix[IdDestino][IdOrigen] = this.poblacionesMatrix[IdOrigen][IdDestino];

         this.HabPoblacion[IdOrigen][IdDestino] = pob;
         this.HabPoblacion[IdDestino][IdOrigen] = pob;
      }
   }

   public long[][] getZonasMatrix() {
      return zonasMatrix;
   }

   public void setZonasMatrix(ArrayList<BaseZonaNatural> zonas) {
      for (BaseZonaNatural zona : zonas) {
         int IdOrigen = zona.getIdOrigen();
         int IdDestino = zona.getIdDestino();
         int KM = zona.getKmZonaNatural();
         int tipo = zona.getTipoZonaNatural() - 1;
         this.zonasMatrix[IdOrigen][IdDestino] = (long) (KM * Parametros.ModificadorZona[tipo] * Parametros.intFactor);
         this.zonasMatrix[IdDestino][IdOrigen] = this.zonasMatrix[IdOrigen][IdDestino];

         this.KMReserva[IdOrigen][IdDestino] = KM;
         this.KMReserva[IdDestino][IdOrigen] = KM;
      }
```

```
    }

    //Construye la matriz de costes teniendo en cuenta el ajuste
    //Introducido en el menu de parámetros. (jSlider)
    public long[][] getCostMatrix() {
        long[][] costMatrix = new long[dim][dim];
        for (int i = 0; i < dim; i++) {
            for (int j = 0; j < dim; j++) {
                costMatrix[i][j] = this.distanciasMatrix[i][j] + Math.round(Parametros.AjusteReservas * this.zonasMatrix[i][j]) +
Math.round(Parametros.AjustePoblaciones * this.poblacionesMatrix[i][j]);
            }
        }
        return costMatrix;
    }

    private static long[][] getMatrixOfDistances(ArrayList<DSSNode> nodesList) {
        int dim = nodesList.size();
        long[][] costsMatrix = new long[dim][dim];

        // Set costsMatrix[i][i] = 0 for all i
        for (int i = 0; i < dim; i++) {
            costsMatrix[i][i] = 0;
        }

        // Set costsMatrix[i][j] = costsMatrix[j][i] = euclideanDistance(i,j)
        for (int i = 1; i < dim; i++) {
            DSSNode iNode = nodesList.get(i);
            float xi = iNode.getX();
            float yi = iNode.getY();

            for (int j = 0; j < i; j++) {
                DSSNode jNode = nodesList.get(j);
                float xj = jNode.getX();
                float yj = jNode.getY();

                double distance = Math.sqrt((xi - xj) * (xi - xj) + (yi - yj) * (yi - yj));
                long intDistance = (long) Math.round(distance * Parametros.intFactor);

                costsMatrix[i][j] = intDistance;
                costsMatrix[j][i] = intDistance;
            }
        }
        return costsMatrix;
    }

    public String getSummary(Solution solucion) {


        String s = "Solution ID: " + solucion.getId() + newline
                + "Total Cost: " + solucion.getCosts() / Parametros.intFactor + newline+newline
                + "Routes: "+newline;


        String details = "|Ruta ID |KM     |Pob    |C.Pob  |Reserv  |C.Reserv|C.Total |"+newline;
        float TDistancia = 0;
        float TPob = 0;
        float TCostePob = 0;
        float TKilometrosReserva = 0;
        float TCosteReserva = 0;
        float TCosteTotal = 0;
        for (Route r : solucion.getRoutes()) {
            s = s + "route " + r.getId() + ": 0->";
            int RutaID = r.getId();
            float Distancia = 0;
            float Pob = 0;
            float CostePob = 0;
            float KilometrosReserva = 0;
            float CosteReserva = 0;
            float CosteTotal = 0;
```

```
            for (Edge e : r.getEdges()) {
                s = s + e.getEnd().getId();
                if (e.getEnd().getId() != 0) {
                    s = s + "->";
                }
                int OID = e.getOrigin().getId();
                int EID = e.getEnd().getId();
                Distancia = Distancia + this.distanciasMatrix[OID][EID];
                Pob = Pob + this.HabPoblacion[OID][EID];
                CostePob = CostePob + this.poblacionesMatrix[OID][EID];
                KilometrosReserva = KilometrosReserva + this.KMReserva[OID][EID];
                CosteReserva = CosteReserva + this.zonasMatrix[OID][EID];
            }
            s = s + newline;
            Distancia = Distancia / Parametros.intFactor;
            CostePob = CostePob / Parametros.intFactor;
            CosteReserva = CosteReserva / Parametros.intFactor;
            CosteTotal = Distancia + CostePob + CosteReserva;
            details = details + "|" + format(RutaID, 8) + format(Distancia, 8) + format(Pob, 8)
                    + format(CostePob, 8) + format(KilometrosReserva, 8)
                    + format(CosteReserva, 8) + format(CosteTotal, 8) + newline;
            TDistancia = TDistancia + Distancia;
            TPob = TPob + Pob;
            TCostePob = TCostePob + CostePob;
            TKilometrosReserva = TKilometrosReserva + KilometrosReserva;
            TCosteReserva = TCosteReserva + CosteReserva;
            TCosteTotal = TCosteTotal + CosteTotal;
        }
        details = details + "|" + format("Total", 8) + format(TDistancia, 8) + format(TPob, 8)
                + format(TCostePob, 8) + format(TKilometrosReserva, 8)
                + format(TCosteReserva, 8) + format(TCosteTotal, 8) + newline;
        return s + newline+newline + details;
    }

    public String format(Object o, int i) {
        String s = o.toString();
        while (s.length() < i) {
            s = s + " ";
        }
        return s + "|";
    }
}
```

Clase DSSNode

```
package Nucleo;

import Algoritmo.Node;
import Interfaces.DrawAble;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;

/**
 *
 * @author Administrador
 */
public class DSSNode extends Node implements DrawAble {

    private String Nombre;
    private int Poblacion;
    private int XEnPanel;
    private int YEnPanel;

    public int getXEnPanel() {
        return XEnPanel;
    }

    public int getYEnPanel() {
        return YEnPanel;
    }

    public DSSNode(int Id, String Nombre, int X, int Y, int Demanda, int Poblacion) {

        super(Id, X, Y, Demanda);
        this.Nombre = "("+Id+")"+ Nombre;
        this.Poblacion = Poblacion;
    }

    public String getNombre() {
        return Nombre;
    }

    public int getPoblacion() {
        return Poblacion;
    }

    public void draw(Graphics g, int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
            int MaxXEscalada, int MaxYEscalada) {
        BufferedImage image = null;
        try {
            String file = System.getProperty("user.dir") + "\\src\\dss\\GUI\\Imagenes\\Small_house.png";
            image = ImageIO.read(new File(file));
        } catch (IOException ex) {
            Logger.getLogger(DSSNode.class.getName()).log(Level.SEVERE, null, ex);
        }

        setCoordenadasEnPanel(AlturaPanel, AnchuraPanel, Escala, DesplazamientoX,
                DesplazamientoY, MaxXEscalada, MaxYEscalada);

        g.drawImage(image, XEnPanel-8, YEnPanel-8, null);
        g.drawString(this.Nombre, XEnPanel + 12, YEnPanel + 5);
    }

    private void setCoordenadasEnPanel(int AlturaPanel, int AnchuraPanel,
            float Escala, int DesplazamientoX, int DesplazamientoY,
```

```
            int MaxX, int MaxY) {

        float MaxXEscalada = ((float) MaxX + DesplazamientoX) * Escala;
        float MaxYEscalada = ((float) MaxY + DesplazamientoY) * Escala;
        //Calculo el valor del X actual tras desplazarla
        float XDesplazada = this.x + DesplazamientoX;
        //Calculo el valor del X actual tras desplazarla y escalarla
        float XEscalada = XDesplazada * Escala;
        //Calculo el valor del X actual tras desplazarla, escalarla y centrar
        //la imagen. Este sera su valor final
        float XCentrada = XEscalada + (AnchuraPanel - MaxXEscalada) / 2;

        //Calculo el valor del Y actual tras desplazarla
        float YDesplazada = this.y + DesplazamientoY;
        //Calculo el valor del Y actual tras desplazarla y escalarla
        float YEscalada = YDesplazada * Escala;
        //Calculo el valor del X actual tras desplazarla, escalarla y centrar
        //la imagen.
        float YCentrada = YEscalada + (AlturaPanel - MaxYEscalada) / 2;
        //Finalmente puesto que en el Jpanel el punto 0,0 corresponde a la
        //Esquina superior izquierda tengo que invertir las Y haciendo que su
        //valor final sea el del alto del panel menos el valor calculado
        //puesto que queremos que el margen se respete tanto por arriba como
        //por abajo, para pintarlo, le restamos a la altura del panel la mitad
        //del margen
//      YCentrada = AlturaPanel - this.Margen / 2 - YCentrada;
        YCentrada = AlturaPanel - YCentrada;

        this.XEnPanel = Math.round(XCentrada)+8;
        this.YEnPanel = Math.round(YCentrada)+8;

    }
}
```

Paquete “dss.GUI”

Clase ConfigDialog

```
package dss.GUI;

import Config.Parametros;
/*
 * ConfigDialog.java
 *
 * Created on 24-ago-2010, 8:58:14
 */

public class ConfigDialog extends javax.swing.JDialog {

    /** Creates new form ConfigDialog */
    public ConfigDialog(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
        loadValues();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel3 = new javax.swing.JPanel();
        jLabel6 = new javax.swing.JLabel();
        Iteraciones = new javax.swing.JTextField();
        jLabel7 = new javax.swing.JLabel();
        BetaMin = new javax.swing.JTextField();
        jLabel8 = new javax.swing.JLabel();
        SolucionesGuardadas = new javax.swing.JTextField();
        jLabel10 = new javax.swing.JLabel();
        IteracionesSplit = new javax.swing.JTextField();
        jLabel11 = new javax.swing.JLabel();
        BetaMax = new javax.swing.JTextField();
        jLabel12 = new javax.swing.JLabel();
        Seed = new javax.swing.JTextField();
        UseLecuyer = new javax.swing.JCheckBox();
        jButton2 = new javax.swing.JButton();
        GuardarButton = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setName("Form"); // NOI18N

        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(dss.GUI.DSSApp.class).getContext().getResourceMap(ConfigDialog.class);
        jPanel3.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("jPanel3.border.title"))); // NOI18N
        jPanel3.setName("jPanel3"); // NOI18N

        jLabel6.setText(resourceMap.getString("jLabel6.text")); // NOI18N
        jLabel6.setName("jLabel6"); // NOI18N

        Iteraciones.setText(resourceMap.getString("Iteraciones.text")); // NOI18N
        Iteraciones.setName("Iteraciones"); // NOI18N

        jLabel7.setText(resourceMap.getString("jLabel7.text")); // NOI18N
        jLabel7.setName("jLabel7"); // NOI18N

        BetaMin.setText(resourceMap.getString("BetaMin.text")); // NOI18N
        BetaMin.setName("BetaMin"); // NOI18N

        jLabel8.setText(resourceMap.getString("jLabel8.text")); // NOI18N
        jLabel8.setName("jLabel8"); // NOI18N
```

```
        SolucionesGuardadas.setText(resourceMap.getString("SolucionesGuardadas.text")); // NOI18N
        SolucionesGuardadas.setName("SolucionesGuardadas"); // NOI18N

        jLabel10.setText(resourceMap.getString("jLabel10.text")); // NOI18N
        jLabel10.setName("jLabel10"); // NOI18N

        IteracionesSplit.setText(resourceMap.getString("IteracionesSplit.text")); // NOI18N
        IteracionesSplit.setName("IteracionesSplit"); // NOI18N

        jLabel11.setText(resourceMap.getString("jLabel11.text")); // NOI18N
        jLabel11.setName("jLabel11"); // NOI18N

        BetaMax.setText(resourceMap.getString("BetaMax.text")); // NOI18N
        BetaMax.setName("BetaMax"); // NOI18N

        jLabel12.setText(resourceMap.getString("jLabel12.text")); // NOI18N
        jLabel12.setName("jLabel12"); // NOI18N

        Seed.setName("Seed"); // NOI18N

        UseLecuyer.setText(resourceMap.getString("UseLecuyer.text")); // NOI18N
        UseLecuyer.setName("UseLecuyer"); // NOI18N

        javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
        jPanel3.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(
            jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel3Layout.createSequentialGroup()
                .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel3Layout.createSequentialGroup()
                        .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel6)
                            .addComponent(jLabel7)
                            .addComponent(jLabel8))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                            .addComponent(SolucionesGuardadas)
                            .addComponent(BetaMin)
                            .addComponent(Iteraciones, javax.swing.GroupLayout.PREFERRED_SIZE, 45,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGap(18, 18, 18)
                        .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addGroup(jPanel3Layout.createSequentialGroup()
                                .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel12)
                                    .addComponent(jLabel11))
                                .addGap(106, 106, 106))
                            .addGroup(jPanel3Layout.createSequentialGroup()
                                .addComponent(jLabel10)
                                .addGap(18, 18, 18)))
                        .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                            .addComponent(IteracionesSplit)
                            .addComponent(Seed, javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(BetaMax, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addComponent(UseLecuyer))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        jPanel3Layout.setVerticalGroup(
            jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel3Layout.createSequentialGroup()
                .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel6)
                    .addComponent(Iteraciones, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel7)
```

```
            .addComponent(BetaMin,      javax.swing.GroupLayout.PREFERRED_SIZE,      javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel11))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel8)
            .addComponent(SolucionesGuardadas,                                javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel12))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(UseLecuyer))
        .addGroup(jPanel3Layout.createSequentialGroup()
          .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(IteracionesSplit,  javax.swing.GroupLayout.PREFERRED_SIZE,  javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabel10))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(BetaMax,      javax.swing.GroupLayout.PREFERRED_SIZE,      javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(Seed,         javax.swing.GroupLayout.PREFERRED_SIZE,         javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
      );

      jButton2.setText(resourceMap.getString("jButton2.text")); // NOI18N
      jButton2.setName("jButton2"); // NOI18N
      jButton2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
          jButton2ActionPerformed(evt);
        }
      });

      GuardarButton.setText(resourceMap.getString("GuardarButton.text")); // NOI18N
      GuardarButton.setName("GuardarButton"); // NOI18N
      GuardarButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
          GuardarButtonActionPerformed(evt);
        }
      });

      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addContainerGap()
          .addComponent(jPanel3,      javax.swing.GroupLayout.PREFERRED_SIZE,      javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
          .addContainerGap(207, Short.MAX_VALUE)
          .addComponent(jButton2)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(GuardarButton)
          .addContainerGap())
      );
      layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addContainerGap()
          .addComponent(jPanel3,      javax.swing.GroupLayout.PREFERRED_SIZE,      javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(GuardarButton)
            .addComponent(jButton2))
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      );
```

```
        pack();
    }// </editor-fold>

    private void GuardarButtonActionPerformed(java.awt.event.ActionEvent evt) {
        Parametros.IteracionesExteriores = Integer.valueOf(this.Iteraciones.getText());
        Parametros.IteracionesSplit = Integer.valueOf(this.IteracionesSplit.getText());
        Parametros.BetaMin = Float.valueOf(this.BetaMin.getText());
        Parametros.BetaMax = Float.valueOf(this.BetaMax.getText());
        Parametros.SolucionesGuardadas = Integer.valueOf(this.SolucionesGuardadas.getText());
        Parametros.Semilla = Integer.valueOf(this.Seed.getText());
        Parametros.UseLecuyer = this.UseLecuyer.isSelected();
        this.dispose();
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        Parametros.setDefaultValues();
        this.loadValues();
    }
    /**
     * @param args the command line arguments
     */
    private javax.swing.JTextField BetaMax;
    private javax.swing.JTextField BetaMin;
    private javax.swing.JButton GuardarButton;
    private javax.swing.JTextField Iteraciones;
    private javax.swing.JTextField IteracionesSplit;
    private javax.swing.JTextField Seed;
    private javax.swing.JTextField SolucionesGuardadas;
    private javax.swing.JCheckBox UseLecuyer;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JPanel jPanel3;
    // End of variables declaration

    private void loadValues() {
        this.Iteraciones.setText(String.valueOf(Parametros.IteracionesExteriores));
        this.IteracionesSplit.setText(String.valueOf(Parametros.IteracionesSplit));
        this.BetaMin.setText(String.valueOf(Parametros.BetaMin));
        this.BetaMax.setText(String.valueOf(Parametros.BetaMax));
        this.SolucionesGuardadas.setText(String.valueOf(Parametros.SolucionesGuardadas));
        this.Seed.setText(String.valueOf(Parametros.Semilla));
        this.UseLecuyer.setSelected(Parametros.UseLecuyer);
    }
}
```

Clase DSSView

```
/*
 * DSSView.java
 */
package dss.GUI;

import Algoritmo.Node;
import Algoritmo.SRGCWSCS;
import Algoritmo.Solution;
import Config.Parametros;
import DSSInOut.DSSInOut;
import Interfaces.DrawAble;
import Nucleo.DSSNode;
import Nucleo.BasePoblacionDePaso;
import Nucleo.BaseZonaNatural;
import Nucleo.DSSCalculator;
import org.jdesktop.application.Action;
import org.jdesktop.application.ResourceMap;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import org.jdesktop.application.TaskMonitor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import javax.swing.Timer;
import javax.swing.Icon;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileNameExtensionFilter;

/**
 * The application's main frame.
 */
public class DSSView extends FrameView {

    private String lastPathUsed;
    private String ficheroPoblaciones;
    private String ficheroNodos;
    private String ficheroSalida;
    private String ficheroZonasNaturales;
    private DSSCalculator Calculator;
    private ArrayList<Solution> soluciones = new ArrayList<Solution>();
    private ArrayList<DSSNode> nodos = new ArrayList<DSSNode>();
    private ArrayList<BasePoblacionDePaso> poblaciones = new ArrayList<BasePoblacionDePaso>();
    private ArrayList<BaseZonaNatural> zonasNaturales = new ArrayList<BaseZonaNatural>();
    private String newline = System.getProperty("line.separator");
    private int currentSolution;

    public DSSView(SingleFrameApplication app) {
        super(app);

        initComponents();

        // status bar initialization - message timeout, idle icon and busy animation, etc
        ResourceMap resourceMap = getResourceMap();
        int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
        messageTimer = new Timer(messageTimeout, new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                statusMessageLabel.setText("");
            }
        });
        messageTimer.setRepeats(false);
```

```
    int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
    for (int i = 0; i < busyIcons.length; i++) {
        busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
    }
    busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {

        public void actionPerformed(ActionEvent e) {
            busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
            statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
        }
    });
    idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
    statusAnimationLabel.setIcon(idleIcon);
    progressBar.setVisible(false);

    // connecting action tasks to status bar via TaskMonitor
    TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
    taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {

        public void propertyChange(java.beans.PropertyChangeEvent evt) {
            String propertyName = evt.getPropertyName();
            if ("started".equals(propertyName)) {
                if (!busyIconTimer.isRunning()) {
                    statusAnimationLabel.setIcon(busyIcons[0]);
                    busyIconIndex = 0;
                    busyIconTimer.start();
                }
                progressBar.setVisible(true);
                progressBar.setIndeterminate(true);
            } else if ("done".equals(propertyName)) {
                busyIconTimer.stop();
                statusAnimationLabel.setIcon(idleIcon);
                progressBar.setVisible(false);
                progressBar.setValue(0);
            } else if ("message".equals(propertyName)) {
                String text = (String) (evt.getNewValue());
                statusMessageLabel.setText((text == null) ? "" : text);
                messageTimer.restart();
            } else if ("progress".equals(propertyName)) {
                int value = (Integer) (evt.getNewValue());
                progressBar.setVisible(true);
                progressBar.setIndeterminate(false);
                progressBar.setValue(value);
            }
        }
    });
}

@Action
public void showAboutBox() {
    if (aboutBox == null) {
        JFrame mainFrame = DSSApp.getApplication().getMainFrame();
        aboutBox = new DSSAboutBox(mainFrame);
        aboutBox.setLocationRelativeTo(mainFrame);
    }
    DSSApp.getApplication().show(aboutBox);
}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    mainPanel = new javax.swing.JPanel();
    jPanel6 = new javax.swing.JPanel();
```

```
        jPanel5 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        CapacidadDelVehiculoField = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        LongitudMaximaDeLaRutaField = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jSlider1 = new javax.swing.JSlider();
        jSlider2 = new javax.swing.JSlider();
        AjuestePoblacionesAfectadasField = new javax.swing.JTextField();
        AjusteZonasNaturalesField = new javax.swing.JTextField();
        jScrollPane1 = new javax.swing.JScrollPane();
        summaryTextArea = new javax.swing.JTextArea();
        MostrarFicheroButton = new javax.swing.JButton();
        avanzarButton = new javax.swing.JButton();
        indexLabel = new javax.swing.JLabel();
        RetrocederButton = new javax.swing.JButton();
        jButton1 = new javax.swing.JButton();
        PainPanel = new dss.GUI.JPaintPanel();
        menuBar = new javax.swing.JMenuBar();
        javax.swing.JMenu fileMenu = new javax.swing.JMenu();
        menuCargarNodos = new javax.swing.JMenuItem();
        MenuCargarPoblaciones = new javax.swing.JMenuItem();
        MenuCargarZonasNaturales = new javax.swing.JMenuItem();
        jMenuItem1 = new javax.swing.JMenuItem();
        javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
        javax.swing.JMenu helpMenu = new javax.swing.JMenu();
        javax.swing.JMenuItem aboutMenuItem = new javax.swing.JMenuItem();
        statusPanel = new javax.swing.JPanel();
        javax.swing.JSeparator statusPanelSeparator = new javax.swing.JSeparator();
        statusMessageLabel = new javax.swing.JLabel();
        statusAnimationLabel = new javax.swing.JLabel();
        progressBar = new javax.swing.JProgressBar();

        org.jdesktop.application.ResourceMap resourceMap =
    org.jdesktop.application.Application.getInstance(dss.GUI.DSSApp.class).getContext().getResourceMap(DSSView.class);
        mainPanel.setBackground(resourceMap.getColor("mainPanel.background")); // NOI18N
        mainPanel.setName("mainPanel"); // NOI18N

        jPanel6.setName("jPanel1"); // NOI18N

        jPanel5.setBorder(javax.swing.BorderFactory.createTitledBorder(resourceMap.getString("jPanel2.border.title"))); // NOI18N
        jPanel5.setName("jPanel2"); // NOI18N

        jLabel2.setText(resourceMap.getString("jLabel2.text")); // NOI18N
        jLabel2.setName("jLabel2"); // NOI18N

        CapacidadDelVehiculoField.setText(resourceMap.getString("CapacidadDelVehiculoField.text")); // NOI18N
        CapacidadDelVehiculoField.setName("CapacidadDelVehiculoField"); // NOI18N

        jLabel3.setText(resourceMap.getString("jLabel3.text")); // NOI18N
        jLabel3.setName("jLabel3"); // NOI18N

        LongitudMaximaDeLaRutaField.setText(resourceMap.getString("LongitudMaximaDeLaRutaField.text")); // NOI18N
        LongitudMaximaDeLaRutaField.setName("LongitudMaximaDeLaRutaField"); // NOI18N

        jLabel4.setText(resourceMap.getString("jLabel4.text")); // NOI18N
        jLabel4.setName("jLabel4"); // NOI18N

        jLabel5.setText(resourceMap.getString("jLabel5.text")); // NOI18N
        jLabel5.setName("jLabel5"); // NOI18N

        jSlider1.setMaximum(10);
        jSlider1.setMinorTickSpacing(1);
        jSlider1.setPaintTicks(true);
        jSlider1.setSnapToTicks(true);
        jSlider1.setValue(5);
        jSlider1.setBorder(javax.swing.BorderFactory.createCompoundBorder());
        jSlider1.setName("jSlider1"); // NOI18N
```

```
        jSlider1.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jSlider1StateChanged(evt);
            }
        });

        jSlider2.setMaximum(10);
        jSlider2.setMinorTickSpacing(1);
        jSlider2.setPaintLabels(true);
        jSlider2.setPaintTicks(true);
        jSlider2.setSnapToTicks(true);
        jSlider2.setValue(5);
        jSlider2.setName("jSlider2"); // NOI18N
        jSlider2.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jSlider2StateChanged(evt);
            }
        });

        AjustePoblacionesAfectadasField.setText(resourceMap.getString("AjustePoblacionesAfectadasField.text")); // NOI18N
        AjustePoblacionesAfectadasField.setName("AjustePoblacionesAfectadasField"); // NOI18N
        AjustePoblacionesAfectadasField.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                AjustePoblacionesAfectadasFieldActionPerformed(evt);
            }
        });

        AjusteZonasNaturalesField.setText(resourceMap.getString("AjusteZonasNaturalesField.text")); // NOI18N
        AjusteZonasNaturalesField.setName("AjusteZonasNaturalesField"); // NOI18N
        AjusteZonasNaturalesField.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                AjusteZonasNaturalesFieldActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel5Layout = new javax.swing.GroupLayout(jPanel5);
        jPanel5.setLayout(jPanel5Layout);
        jPanel5Layout.setHorizontalGroup(
            jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel5Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 138, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                    .addComponent(CapacidadDelVehiculoField)
                    .addComponent(LongitudMaximaDeLaRutaField, javax.swing.GroupLayout.DEFAULT_SIZE, 45, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel5)
                    .addComponent(jLabel4))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(AjustePoblacionesAfectadasField, javax.swing.GroupLayout.PREFERRED_SIZE, 18,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(AjusteZonasNaturalesField, javax.swing.GroupLayout.PREFERRED_SIZE, 18,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jSlider1, javax.swing.GroupLayout.DEFAULT_SIZE, 122, Short.MAX_VALUE)
                    .addComponent(jSlider2, javax.swing.GroupLayout.DEFAULT_SIZE, 122, Short.MAX_VALUE)))
        );
        jPanel5Layout.setVerticalGroup(
            jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel5Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
                .addComponent(CapacidadDelVehiculoField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel5)
                .addComponent(AjustePoblacionesAfectadasField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel2))
            .addComponent(jSlider1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGap(12, 12, 12)
          .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(LongitudMaximaDeLaRutaField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(AjusteZonasNaturalesField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(jLabel4)
              .addComponent(jLabel3))
            .addComponent(jSlider2, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );

    jScrollPane1.setName("jScrollPane1"); // NOI18N

    summaryTextArea.setColumns(20);
    summaryTextArea.setEditable(false);
    summaryTextArea.setRows(5);
    summaryTextArea.setText(resourceMap.getString("summaryTextArea.text")); // NOI18N
    summaryTextArea.setName("summaryTextArea"); // NOI18N
    jScrollPane1.setViewportView(summaryTextArea);

    javax.swing.ActionMap actionMap =
org.jdesktop.application.Application.getInstance(dss.GUI.DSSApp.class).getContext().getActionMap(DSSView.class, this);
    MostrarFicheroButton.setAction(actionMap.get("MostrarFicheroAction")); // NOI18N
    MostrarFicheroButton.setText(resourceMap.getString("MostrarFicheroButton.text")); // NOI18N
    MostrarFicheroButton.setName("MostrarFicheroButton"); // NOI18N

    avanzarButton.setText(resourceMap.getString("avanzarButton.text")); // NOI18N
    avanzarButton.setName("avanzarButton"); // NOI18N
    avanzarButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        avanzarButtonActionPerformed(evt);
      }
    });

    indexLabel.setText(resourceMap.getString("indexLabel.text")); // NOI18N
    indexLabel.setName("indexLabel"); // NOI18N

    RetrocederButton.setText(resourceMap.getString("RetrocederButton.text")); // NOI18N
    RetrocederButton.setName("RetrocederButton"); // NOI18N
    RetrocederButton.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        RetrocederButtonActionPerformed(evt);
      }
    });

    jButton1.setText(resourceMap.getString("jButton1.text")); // NOI18N
    jButton1.setName("jButton1"); // NOI18N
    jButton1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel6Layout = new javax.swing.GroupLayout(jPanel6);
    jPanel6.setLayout(jPanel6Layout);
    jPanel6Layout.setHorizontalGroup(
      jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel6Layout.createSequentialGroup()
```

```
            .addContainerGap()
            .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 520, Short.MAX_VALUE)
              .addComponent(jPanel5,  javax.swing.GroupLayout.DEFAULT_SIZE,  javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
              .addGroup(jPanel6Layout.createSequentialGroup()
                .addComponent(MostrarFicheroButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jButton1)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 174, Short.MAX_VALUE)
                .addComponent(RetrocederButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(indexLabel)
                .addGap(12, 12, 12)
                .addComponent(avanzarButton)))
            .addContainerGap())
        );
        jPanel6Layout.setVerticalGroup(
          jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(jPanel6Layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jPanel5,  javax.swing.GroupLayout.PREFERRED_SIZE,  javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel6Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(MostrarFicheroButton)
              .addComponent(jButton1)
              .addComponent(avanzarButton)
              .addComponent(RetrocederButton)
              .addComponent(indexLabel))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 300, Short.MAX_VALUE)
            .addContainerGap())
        );

        PainPanel.setBackground(resourceMap.getColor("PainPanel.background")); // NOI18N
        PainPanel.setName("PainPanel"); // NOI18N

        javax.swing.GroupLayout PainPanelLayout = new javax.swing.GroupLayout(PainPanel);
        PainPanel.setLayout(PainPanelLayout);
        PainPanelLayout.setHorizontalGroup(
          PainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGap(0, 464, Short.MAX_VALUE)
        );
        PainPanelLayout.setVerticalGroup(
          PainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGap(0, 492, Short.MAX_VALUE)
        );

        javax.swing.GroupLayout mainPanelLayout = new javax.swing.GroupLayout(mainPanel);
        mainPanel.setLayout(mainPanelLayout);
        mainPanelLayout.setHorizontalGroup(
          mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, mainPanelLayout.createSequentialGroup()
            .addContainerGap()
            .addComponent(PainPanel,  javax.swing.GroupLayout.DEFAULT_SIZE,  javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(jPanel6,  javax.swing.GroupLayout.PREFERRED_SIZE,  javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
        );
        mainPanelLayout.setVerticalGroup(
          mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, mainPanelLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
              .addComponent(PainPanel,  javax.swing.GroupLayout.Alignment.LEADING,  javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
            .addComponent(jPanel6, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addContainerGap())
        );

        menuBar.setName("menuBar"); // NOI18N

        fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N
        fileMenu.setName("fileMenu"); // NOI18N
        fileMenu.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                fileMenuActionPerformed(evt);
            }
        });

        menuCargarNodos.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_N,
java.awt.event.InputEvent.CTRL_MASK));
        menuCargarNodos.setText(resourceMap.getString("menuCargarNodos.text")); // NOI18N
        menuCargarNodos.setName("menuCargarNodos"); // NOI18N
        menuCargarNodos.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                menuCargarNodosActionPerformed(evt);
            }
        });
        fileMenu.add(menuCargarNodos);

        MenuCargarPoblaciones.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_P,
java.awt.event.InputEvent.CTRL_MASK));
        MenuCargarPoblaciones.setText(resourceMap.getString("MenuCargarPoblaciones.text")); // NOI18N
        MenuCargarPoblaciones.setName("MenuCargarPoblaciones"); // NOI18N
        MenuCargarPoblaciones.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                MenuCargarPoblacionesActionPerformed(evt);
            }
        });
        fileMenu.add(MenuCargarPoblaciones);

        MenuCargarZonasNaturales.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_R,
java.awt.event.InputEvent.CTRL_MASK));
        MenuCargarZonasNaturales.setText(resourceMap.getString("MenuCargarZonasNaturales.text")); // NOI18N
        MenuCargarZonasNaturales.setName("MenuCargarZonasNaturales"); // NOI18N
        MenuCargarZonasNaturales.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                MenuCargarZonasNaturalesActionPerformed(evt);
            }
        });
        fileMenu.add(MenuCargarZonasNaturales);

        jMenuItem1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_X,
java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem1.setText(resourceMap.getString("jMenuItem1.text")); // NOI18N
        jMenuItem1.setName("jMenuItem1"); // NOI18N
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem1ActionPerformed(evt);
            }
        });
        fileMenu.add(jMenuItem1);

        exitMenuItem.setAction(actionMap.get("quit")); // NOI18N
        exitMenuItem.setName("exitMenuItem"); // NOI18N
        fileMenu.add(exitMenuItem);

        menuBar.add(fileMenu);

        helpMenu.setText(resourceMap.getString("helpMenu.text")); // NOI18N
        helpMenu.setName("helpMenu"); // NOI18N

        aboutMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N
```

```
        aboutMenuItem.setName("aboutMenuItem"); // NOI18N
        helpMenu.add(aboutMenuItem);

        menuBar.add(helpMenu);
        statusPanel.setName("statusPanel"); // NOI18N
        statusPanelSeparator.setName("statusPanelSeparator"); // NOI18N
        statusMessageLabel.setName("statusMessageLabel"); // NOI18N
        statusAnimationLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        statusAnimationLabel.setName("statusAnimationLabel"); // NOI18N
        progressBar.setName("progressBar"); // NOI18N
        javax.swing.GroupLayout statusPanelLayout = new javax.swing.GroupLayout(statusPanel);
        statusPanel.setLayout(statusPanelLayout);
        statusPanelLayout.setHorizontalGroup(
            statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(statusPanelSeparator, javax.swing.GroupLayout.DEFAULT_SIZE, 1034, Short.MAX_VALUE)
            .addGroup(statusPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(statusMessageLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 864, Short.MAX_VALUE)
                .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(statusAnimationLabel)
                .addContainerGap())
        );
        statusPanelLayout.setVerticalGroup(
            statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(statusPanelLayout.createSequentialGroup()
                .addComponent(statusPanelSeparator, javax.swing.GroupLayout.PREFERRED_SIZE, 2,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(statusMessageLabel)
                    .addComponent(statusAnimationLabel)
                    .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(3, 3, 3))
        );

        setComponent(mainPanel);
        setMenuBar(menuBar);
        setStatusBar(statusPanel);
    }// </editor-fold>

    private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
        ConfigDialog CD = new ConfigDialog(this.getFrame(), true);
        CD.setLocationRelativeTo(null);
        CD.setVisible(true);
    }

    private void menuCargarNodosActionPerformed(java.awt.event.ActionEvent evt) {
        String ruta = getRutaFichero();
        if (ruta != null) {
            ficheroNodos = ruta;
        }

        this.nodos = DSSInOut.getNodesList(ficheroNodos);
        //Se pasa al DSSCalculator el ArrayList de nodos.
        //Crea la matriz de distancias.
        Calculator = new DSSCalculator(nodos);
        pintarNodos(nodos);

    }

    private void MenuCargarPoblacionesActionPerformed(java.awt.event.ActionEvent evt) {

        if (this.Calculator != null) {
            String ruta = getRutaFichero();
```

```
        if (ruta != null) {
           ficheroPoblaciones = ruta;
        }

        //Guarda en un ArrayList los datos de poblaciones
        poblaciones = DSSInOut.getPoblacionesList(ficheroPoblaciones);
        //Se pasa al DssCalculator el ArrayList de poblaciones.
        //Crea la matriz de poblaciones
        Calculator.setPoblacionesMatrix(poblaciones);
        //Pintar
        pintarPoblaciones(poblaciones);
     } else {
        JOptionPane.showMessageDialog(null, "Primero debbe cargar un fichero de nodos", "Falta fichero de nodos",
JOptionPane.WARNING_MESSAGE);
     }

  }

  private void MenuCargarZonasNaturalesActionPerformed(java.awt.event.ActionEvent evt) {
     if (this.Calculator != null) {
        String ruta = getRutaFichero();

        if (ruta != null) {
           ficheroZonasNaturales = ruta;
        }
        zonasNaturales = DSSInOut.getZonaNaturalList(ficheroZonasNaturales);
        //Se pasa al DssCalculator el ArrayList de zonas naturales.
        //Crea la matriz de zonas naturales
        Calculator.setZonasMatrix(zonasNaturales);
        //Pintar
        pintarZonasNaturales(zonasNaturales);
     } else {
        JOptionPane.showMessageDialog(null, "Primero debbe cargar un fichero de nodos", "Falta fichero de nodos",
JOptionPane.WARNING_MESSAGE);
     }
  }

  private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
     String valor = String.valueOf(jSlider1.getValue());
     AjuestePoblacionesAfectadasField.setText(valor);
  }

  private void jSlider2StateChanged(javax.swing.event.ChangeEvent evt) {
     String valor = String.valueOf(jSlider2.getValue());
     AjusteZonasNaturalesField.setText(valor);
  }

  private void AjuestePoblacionesAfectadasFieldActionPerformed(java.awt.event.ActionEvent evt) {
     jSlider1.setValue(Integer.parseInt(AjuestePoblacionesAfectadasField.getText()));
  }

  private void AjusteZonasNaturalesFieldActionPerformed(java.awt.event.ActionEvent evt) {
     jSlider2.setValue(Integer.parseInt(AjusteZonasNaturalesField.getText()));
  }

  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
     this.soluciones.clear();
     if (this.ficheroNodos == null) {
        JOptionPane.showMessageDialog(null, "Primero debbe cargar un fichero de nodos", "Falta fichero de nodos",
JOptionPane.WARNING_MESSAGE);
     } else {

        CargarParametros();
        ArrayList<Node> nodeList = new ArrayList<Node>();
        copiarNodos(nodeList, nodos);
        long[][] costMat = Calculator.getCostMatrix();
        soluciones = SRGCWSCS.getSolutions(nodeList, costMat);
        ficheroSalida = ficheroNodos.substring(0, ficheroNodos.length() - 4) + timeStamp() + ".txt";
        guardarSolucionesEnFichero();
```

```
        this.currentSolution = 0;
        mostrarSolucion(currentSolution);

    }
}

private void fileMenuActionPerformed(java.awt.event.ActionEvent evt) {
}

private void RetrocederButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (currentSolution > 0) {
        currentSolution--;
        mostrarSolucion(currentSolution);
    }
}

private void avanzarButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if (currentSolution < soluciones.size()-1) {
        currentSolution++;
        mostrarSolucion(currentSolution);
    }
}

@Action
public void MostrarFicheroAction() {
    Process pr;
    String comando = "notepad " + ficheroSalida;
    try {
        pr = Runtime.getRuntime().exec(comando);
    } catch (Exception ex) {
        System.out.println("Ha ocurrido un error al ejecutar el comando. Error: " + ex);
    }

}
// Variables declaration - do not modify
private javax.swing.JTextField AjuestePoblacionesAfectadasField;
private javax.swing.JTextField AjusteZonasNaturalesField;
private javax.swing.JTextField CapacidadDelVehiculoField;
private javax.swing.JTextField LongitudMaximaDeLaRutaField;
private javax.swing.JMenuItem MenuCargarPoblaciones;
private javax.swing.JMenuItem MenuCargarZonasNaturales;
private javax.swing.JButton MostrarFicheroButton;
private dss.GUI.JPaintPanel PainPanel;
private javax.swing.JButton RetrocederButton;
private javax.swing.JButton avanzarButton;
private javax.swing.JLabel indexLabel;
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JPanel jPanel5;
private javax.swing.JPanel jPanel6;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JSlider jSlider1;
private javax.swing.JSlider jSlider2;
private javax.swing.JPanel mainPanel;
private javax.swing.JMenuBar menuBar;
private javax.swing.JMenuItem menuCargarNodos;
private javax.swing.JProgressBar progressBar;
private javax.swing.JLabel statusAnimationLabel;
private javax.swing.JLabel statusMessageLabel;
private javax.swing.JPanel statusPanel;
private javax.swing.JTextArea summaryTextArea;
// End of variables declaration
private final Timer messageTimer;
private final Timer busyIconTimer;
private final Icon idleIcon;
```

```
private final Icon[] busyIcons = new Icon[15];
private int busyIconIndex = 0;
private JDialog aboutBox;

private String getRutaFichero() {
   File f = new File(System.getProperty("user.dir") + "\\instances");
   if ((lastPathUsed == null) && (f.exists())) {
      lastPathUsed = f.getPath();
   } else {
      f = new File("C:\\Users\\koldo\\Desktop");
      if ((lastPathUsed == null) && (f.exists())) {
         lastPathUsed = f.getPath();
      }
   }
   JFileChooser FC = new JFileChooser(lastPathUsed);
   FC.setFileSelectionMode(JFileChooser.FILES_ONLY);
   FileNameExtensionFilter filter = new FileNameExtensionFilter("text files", "txt");
   FC.setFileFilter(filter);

   int returnVal = FC.showSaveDialog(null);
   if (returnVal == JFileChooser.APPROVE_OPTION) {
      File file = FC.getSelectedFile();
      lastPathUsed = file.getParent();
      return file.getPath();
   } else {
      return null;
   }
}

private void pintarNodos(ArrayList<DSSNode> nodos) {
   this.PainPanel.clear();
   Solution solucionInicial = new Solution();
   ArrayList<Node> nodes = new ArrayList<Node>();
   this.copiarNodos(nodes, nodos);
   solucionInicial.setNodos(nodes);
   PainPanel.Solucion = solucionInicial;
   PainPanel.repaint();
}

private void pintarPoblaciones(ArrayList<BasePoblacionDePaso> poblaciones) {
   for (DrawAble d : poblaciones) {
      this.PainPanel.addToDraw(d);
   }
   PainPanel.repaint();
}

private void pintarZonasNaturales(ArrayList<BaseZonaNatural> zonasNaturales) {
   for (DrawAble d : zonasNaturales) {
      this.PainPanel.addToDraw(d);
   }
   PainPanel.repaint();
}

private void CargarParametros() {
   Parametros.CapacidadDelVehiculo = Integer.valueOf(this.CapacidadDelVehiculoField.getText());
   Parametros.setLongitudMaximaDeLaRuta(Integer.valueOf(this.LongitudMaximaDeLaRutaField.getText()));
   Parametros.AjustePoblaciones = Integer.valueOf(this.AjuestePoblacionesAfectadasField.getText());
   Parametros.AjusteReservas = Integer.valueOf(this.AjusteZonasNaturalesField.getText());


}

private void copiarNodos(ArrayList<Node> nodeList, ArrayList<DSSNode> nodos) {
   for (Node n : nodos) {
      nodeList.add(n);
   }
}

private void pintarSolucion(int i) {
```

```
        PainPanel.Solucion = this.soluciones.get(i);
        PainPanel.repaint();
    }

    private void mostrarSolucion(int i) {
        this.indexLabel.setText((currentSolution+1) + " de " + soluciones.size());
        mostrarResumenSolucion(i);
        pintarSolucion(i);
    }

    private void mostrarResumenSolucion(int i) {
        Solution Sol = this.soluciones.get(i);
        String s = this.Calculator.getSummary(Sol);
        this.summaryTextArea.setText(s);
    }

    private String timeStamp() {
        String DATE_FORMAT_NOW = "yyyy-MM-dd HH:mm:ss";
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT_NOW);
        return sdf.format(cal.getTime()).replaceAll(" ", "").replaceAll("-", "").replaceAll(":", "");
    }

    private void guardarSolucionesEnFichero() {
        String text = "";
        String separador = "**********************************************************************";
        int i = 0;
        for (Solution s : this.soluciones) {
            text = text + newline + newline + separador + newline
                    + "Solucion: " + i + newline
                    + separador + newline
                    + this.Calculator.getSummary(s);
            i++;
        }
        DSSInOut.saveToFile(text, ficheroSalida);
    }
}
```

Clase JPaintPanel

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package dss.GUI;

import Algoritmo.Node;
import Algoritmo.Solution;
import Interfaces.DrawAble;
import java.awt.Graphics;
import java.util.ArrayList;
import javax.swing.JPanel;

public class JPaintPanel extends JPanel {

   public ArrayList<DrawAble> ObjectsToDraw = new ArrayList<DrawAble>();
   public Solution Solucion = new Solution();
   int Margen = 40;

   public void addToDraw(DrawAble d) {
      ObjectsToDraw.add(d);
   }

   public void clear() {
      ObjectsToDraw.clear();
   }

   @Override
   protected void paintComponent(Graphics g) {
      super.paintComponent(g);
      //En primer lugar calculamos la escala y el desplazamiento que son
      //Comunes para todos los nodos
      float[] param = getEscalaYDesplazamiento();

      for (DrawAble d : ObjectsToDraw) {
         //Para cada nodo calculamos su posicion
         d.draw(g, this.getHeight()-Margen,this.getWidth()- Margen, param[0]
               , (int)param[1], (int)param[2],(int)param[3],(int)param[4]);
      }
      this.Solucion.draw(g, this.getHeight()-Margen,this.getWidth()- Margen, param[0]
               , (int)param[1], (int)param[2],(int)param[3],(int)param[4]);
   }

   private float[] getEscalaYDesplazamiento() {

      int MaxX = Integer.MIN_VALUE;
      int MinX = Integer.MAX_VALUE;

      int MaxY = Integer.MIN_VALUE;
      int MinY = Integer.MAX_VALUE;

      int AnchuraPanel = this.getWidth();
      int AlturaPanel = this.getHeight() - Margen;


      //Hallamos el mayor y menor valor de X e Y
      for (Node d : Solucion.getNodes()) {
         if (d.getX() > MaxX) {
            MaxX = d.getX();
         }
         if (d.getY() > MaxY) {
            MaxY = d.getY();
         }
         if (d.getX() < MinX) {
            MinX = d.getX();
         }
         if (d.getY() < MinY) {
```

```
                MinY = d.getY();
            }
        }

        //Calculamos el desplazamiento para encuadrar el dibujo en el origen
        //Tomaremos el negativo del minimo de manera que si es negativo le
        //sumaremos su valor para que quede en el 0 y si es positivo se lo
        //restaremos
        int DesplazamientoX = -MinX;
        int DesplazamientoY = -MinY;

        //Calculamos la escala para aprovechar todo el espacio
        //Tomamos el menor de las escalas en X e Y utilizandola en las dos
        //coordenadas la misma para no deformar la imagen
        float Escala = Math.min((float)AnchuraPanel / (MaxX + DesplazamientoX), (float)AlturaPanel / (MaxY + DesplazamientoY));


        float[] param = new float[5];
        param[0] = Escala;
        param[1] = DesplazamientoX;
        param[2] = DesplazamientoY;
        param[3] = MaxX;
        param[4] = MaxY;

        return param;


    }

}
```

Anexo 2. Soluciones de las pruebas realizadas

Prueba 1

```
**************************************************************
Solucion: 0
**************************************************************
Solution ID: 14034
Total Cost: 583.7029
Routes:
route 49: 0->2->5->12->9->1->6->7->0
route 269672: 0->3->0
route 142: 0->14->4->10->8->11->0
route 269684: 0->15->16->20->24->17->0
route 37574: 0->18->19->21->22->23->13->0

|Ruta ID |KM     |Pob    |C.Pob  |Reserv  |C.Reserv|C.Total |
|49     |136.6417|0.0    |0.0    |0.0    |0.0    |136.6417|
|269672  |10.7704 |0.0    |0.0    |0.0    |0.0    |10.7704 |
|142    |71.445  |0.0    |0.0    |0.0    |0.0    |71.445  |
|269684  |170.1064|0.0    |0.0    |0.0    |0.0    |170.1064|
|37574   |194.7394|0.0    |0.0    |0.0    |0.0    |194.7394|
|Total   |583.7029|0.0    |0.0    |0.0    |0.0    |583.7029|

**************************************************************
Solucion: 1
**************************************************************
Solution ID: 86
Total Cost: 599.6983
Routes:
route 1582: 0->3->22->21->19->17->0
route 1587: 0->2->5->15->16->20->24->18->0
route 1590: 0->9->12->1->6->7->0
route 1599: 0->11->0
route 98: 0->8->10->14->4->13->23->0

|Ruta ID |KM     |Pob    |C.Pob  |Reserv  |C.Reserv|C.Total |
|1582   |173.2489|0.0    |0.0    |0.0    |0.0    |173.2489|
|1587   |180.7948|0.0    |0.0    |0.0    |0.0    |180.7948|
|1590   |114.3963|0.0    |0.0    |0.0    |0.0    |114.3963|
|1599   |12.8062 |0.0    |0.0    |0.0    |0.0    |12.8062 |
|98     |118.4521|0.0    |0.0    |0.0    |0.0    |118.4521|
|Total   |599.6983|0.0    |0.0    |0.0    |0.0    |599.6983|

**************************************************************
Solucion: 2
**************************************************************
Solution ID: 3
Total Cost: 620.5349

Routes:
route 2: 0->2->5->9->12->1->6->7->0
route 3: 0->3->17->16->18->0
route 8: 0->8->10->4->14->13->23->0
route 11: 0->11->0
route 15: 0->15->20->24->19->21->22->0

|Ruta ID |KM     |Pob    |C.Pob  |Reserv  |C.Reserv|C.Total |
|2      |139.045 |0.0    |0.0    |0.0    |0.0    |139.045 |
|3      |100.9302|0.0    |0.0    |0.0    |0.0    |100.9302|
|8      |120.5449|0.0    |0.0    |0.0    |0.0    |120.5449|
|11     |12.8062 |0.0    |0.0    |0.0    |0.0    |12.8062 |
|15     |247.2086|0.0    |0.0    |0.0    |0.0    |247.2086|
|Total   |620.5349|0.0    |0.0    |0.0    |0.0    |620.5349|
```

Prueba 2

Ajuste Poblaciones = 1

*****
Solucion: 0
*****
Solution ID: 2707
Total Cost: 590.986

Routes:
route 51558: 0->15->16->20->24->17->0
route 37515: 0->18->19->21->22->23->13->0
route 51564: 0->3->0
route 3079: 0->2->5->12->9->1->6->7->0
route 1647: 0->10->14->4->8->11->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|51558 |170.1064|0.0 |0.0 |0.0 |0.0 |170.1064|
|37515 |194.7394|1000.0 |4.0 |0.0 |0.0 |198.7394|
|51564 |10.7704 |0.0 |0.0 |0.0 |0.0 |10.7704 |
|3079 |136.6417|850.0 |3.0 |0.0 |0.0 |139.6417|
|1647 |71.7281 |0.0 |0.0 |0.0 |0.0 |71.7281 |
|Total |583.98596|1850.0 |7.0 |0.0 |0.0 |590.98596|

*****
Solucion: 1
*****
Solution ID: 636
Total Cost: 602.6983

Routes:
route 12497: 0->9->12->1->6->7->0
route 12505: 0->8->10->14->4->13->23->0
route 12506: 0->11->0
route 9796: 0->2->5->15->16->20->24->18->0
route 5805: 0->3->22->21->19->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|12497 |114.3963|0.0 |0.0 |0.0 |0.0 |114.3963|
|12505 |118.4521|0.0 |0.0 |0.0 |0.0 |118.4521|
|12506 |12.8062 |0.0 |0.0 |0.0 |0.0 |12.8062 |
|9796 |180.7948|850.0 |3.0 |0.0 |0.0 |183.7948|
|5805 |173.2489|0.0 |0.0 |0.0 |0.0 |173.2489|
|Total |599.6983|850.0 |3.0 |0.0 |0.0 |602.6983|

*****
Solucion: 2
*****
Solution ID: 57349
Total Cost: 606.1651

Routes:
route 1677: 0->10->14->4->1->8->0
route 1099875: 0->3->17->24->20->16->0
route 1099876: 0->11->0
route 37515: 0->18->19->21->22->23->13->0
route 1673: 0->2->15->5->12->9->6->7->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
1677	97.0254	0.0	0.0	0.0	0.0	97.0254
1099875	133.279	0.0	0.0	0.0	0.0	133.279
1099876	12.8062	0.0	0.0	0.0	0.0	12.8062
37515	194.7394	1000.0	4.0	0.0	0.0	198.7394
1673	164.3151	0.0	0.0	0.0	0.0	164.3151
Total	602.1651	1000.0	4.0	0.0	0.0	606.1651

****
Solucion: 3
****
Solution ID: 154
Total Cost: 610.5199

Routes:
route 3073: 0->3->0
route 392: 0->13->23->22->21->17->0
route 3079: 0->2->5->12->9->1->6->7->0
route 1647: 0->10->14->4->8->11->0
route 395: 0->15->16->20->24->19->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3073	10.7704	0.0	0.0	0.0	0.0	10.7704
392	187.4679	0.0	0.0	0.0	0.0	187.4679
3079	136.6417	850.0	3.0	0.0	0.0	139.6417
1647	71.7281	0.0	0.0	0.0	0.0	71.7281
395	192.9118	2000.0	8.0	0.0	0.0	200.9118
Total	599.5199	2850.0	11.0	0.0	0.0	610.5199

****
Solucion: 4
****
Solution ID: 3
Total Cost: 649.2847

Routes:
route 2: 0->2->5->9->12->1->6->0
route 3: 0->3->23->13->0
route 7: 0->10->4->14->7->8->11->0
route 15: 0->15->20->24->19->21->22->0
route 16: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total

2	139.0425	850.0	3.0	0.0	0.0	142.0425
3	79.7286	0.0	0.0	0.0	0.0	79.7286
7	76.4931	0.0	0.0	0.0	0.0	76.4931
15	247.2086	1000.0	4.0	0.0	0.0	251.2086
16	99.8119	0.0	0.0	0.0	0.0	99.8119
Total	642.28467	1850.0	7.0	0.0	0.0	649.28467

Ajuste Poblaciones = 3

****
Solucion: 0
****
Solution ID: 180668
Total Cost: 603.6657

Routes:
route 3468839: 0->3->11->0
route 3411355: 0->10->14->4->1->6->8->0
route 3468852: 0->7->9->12->5->15->2->0
route 3423849: 0->17->24->20->16->18->0
route 3410969: 0->13->23->22->21->19->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3468839	23.4502	0.0	0.0	0.0	0.0	23.4502
3411355	97.3134	0.0	0.0	0.0	0.0	97.3134
3468852	157.898	0.0	0.0	0.0	0.0	157.898
3423849	135.0312	0.0	0.0	0.0	0.0	135.0312
3410969	189.9729	0.0	0.0	0.0	0.0	189.9729
Total	603.6657	0.0	0.0	0.0	0.0	603.6657

****
Solucion: 1
****
Solution ID: 184958
Total Cost: 604.986

Routes:
route 3407705: 0->2->5->12->9->1->6->7->0
route 3553431: 0->3->0
route 3553438: 0->10->14->4->8->11->0
route 3553443: 0->15->16->20->24->17->0
route 3553446: 0->18->19->21->22->23->13->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3407705	136.6417	850.0	3.0	0.0	0.0	139.6417
3553431	10.7704	0.0	0.0	0.0	0.0	10.7704
3553438	71.7281	0.0	0.0	0.0	0.0	71.7281
3553443	170.1064	0.0	0.0	0.0	0.0	170.1064
3553446	194.7394	1000.0	4.0	0.0	0.0	198.7394
Total	583.98596	1850.0	7.0	0.0	0.0	590.98596

****
Solucion: 2
****
Solution ID: 177619
Total Cost: 605.1077

Routes:
route 3407599: 0->2->15->16->20->24->18->0
route 3407600: 0->3->22->21->19->17->0
route 3407985: 0->8->10->14->4->13->23->0
route 3407602: 0->5->12->9->1->6->7->0
route 3407608: 0->11->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3407599	165.1787	0.0	0.0	0.0	0.0	165.1787
3407600	173.2489	0.0	0.0	0.0	0.0	173.2489
3407985	118.4521	0.0	0.0	0.0	0.0	118.4521
3407602	135.4218	0.0	0.0	0.0	0.0	135.4218
3407608	12.8062	0.0	0.0	0.0	0.0	12.8062
Total	605.1077	0.0	0.0	0.0	0.0	605.1077

****
Solucion: 3
****
Solution ID: 177591
Total Cost: 625.1329

Routes:
route 3407554: 0->9->12->1->6->8->0
route 3407555: 0->2->5->15->16->20->24->18->0
route 3407556: 0->3->22->21->19->17->0
route 3407560: 0->7->14->4->10->13->23->0
route 3407564: 0->11->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3407554	114.5383	0.0	0.0	0.0	0.0	114.5383
3407555	180.7948	850.0	3.0	0.0	0.0	183.7948
3407556	173.2489	0.0	0.0	0.0	0.0	173.2489
3407560	134.7447	0.0	0.0	0.0	0.0	134.7447
3407564	12.8062	0.0	0.0	0.0	0.0	12.8062
Total	616.13293	850.0	3.0	0.0	0.0	619.13293

Ajuste Poblaciones = 5

****
Solucion: 0
****
Solution ID: 74113
Total Cost: 603.6657

Routes:
route 1421002: 0->3->11->0
route 1421004: 0->2->15->5->12->9->7->0
route 1419957: 0->10->14->4->1->6->8->0
route 1419845: 0->17->24->20->16->18->0
route 1418426: 0->13->23->22->21->19->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|1421002 |23.4502 |0.0 |0.0 |0.0 |0.0 |23.4502 |
|1421004 |157.898 |0.0 |0.0 |0.0 |0.0 |157.898 |
|1419957 |97.3134 |0.0 |0.0 |0.0 |0.0 |97.3134 |
|1419845 |135.0312|0.0 |0.0 |0.0 |0.0 |135.0312|
|1418426 |189.9729|0.0 |0.0 |0.0 |0.0 |189.9729|
|Total |603.6657|0.0 |0.0 |0.0 |0.0 |603.6657|

****
Solucion: 1
****
Solution ID: 74608
Total Cost: 605.1077

Routes:
route 1430543: 0->3->22->21->19->17->0
route 1430548: 0->8->10->14->4->13->23->0
route 1430551: 0->5->12->9->1->6->7->0
route 1430552: 0->11->0
route 1427185: 0->18->24->20->16->15->2->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|1430543 |173.2489|0.0 |0.0 |0.0 |0.0 |173.2489|
|1430548 |118.4521|0.0 |0.0 |0.0 |0.0 |118.4521|
|1430551 |135.4218|0.0 |0.0 |0.0 |0.0 |135.4218|
|1430552 |12.8062 |0.0 |0.0 |0.0 |0.0 |12.8062 |
|1427185 |165.1787|0.0 |0.0 |0.0 |0.0 |165.1787|
|Total |605.1077|0.0 |0.0 |0.0 |0.0 |605.1077|

****
Solucion: 2
****
Solution ID: 73980
Total Cost: 679.3175

Routes:

route 1417999: 0->2->15->5->12->9->0
route 1418000: 0->3->22->21->19->17->0
route 1418003: 0->7->6->14->4->1->10->0
route 1418005: 0->8->11->0
route 1418015: 0->16->20->24->18->23->13->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
1417999	156.7141	0.0	0.0	0.0	0.0	156.7141
1418000	173.2489	0.0	0.0	0.0	0.0	173.2489
1418003	131.2658	0.0	0.0	0.0	0.0	131.2658
1418005	28.1078	0.0	0.0	0.0	0.0	28.1078
1418015	189.9809	0.0	0.0	0.0	0.0	189.9809
Total	679.3175	0.0	0.0	0.0	0.0	679.3175

Ajuste Poblaciones = 10

****
Solucion: 0
****
Solution ID: 126978
Total Cost: 603.6657

Routes:
route 2437211: 0->3->11->0
route 2437216: 0->10->14->4->1->6->8->0
route 2431980: 0->13->23->22->21->19->0
route 2436292: 0->7->9->12->5->15->2->0
route 2431984: 0->17->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2437211	23.4502	0.0	0.0	0.0	0.0	23.4502
2437216	97.3134	0.0	0.0	0.0	0.0	97.3134
2431980	189.9729	0.0	0.0	0.0	0.0	189.9729
2436292	157.898	0.0	0.0	0.0	0.0	157.898
2431984	135.0312	0.0	0.0	0.0	0.0	135.0312
Total	603.6657	0.0	0.0	0.0	0.0	603.6657

****
Solucion: 1
****
Solution ID: 134262
Total Cost: 613.8653

Routes:
route 2576760: 0->11->0
route 2500490: 0->23->13->4->14->7->8->0
route 2443018: 0->3->22->21->19->17->0
route 2433297: 0->2->15->16->20->24->18->0
route 2434581: 0->5->12->9->1->6->10->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2576760	12.8062	0.0	0.0	0.0	0.0	12.8062
2500490	123.0534	0.0	0.0	0.0	0.0	123.0534
2443018	173.2489	0.0	0.0	0.0	0.0	173.2489
2433297	165.1787	0.0	0.0	0.0	0.0	165.1787
2434581	139.5781	0.0	0.0	0.0	0.0	139.5781
Total	613.86536	0.0	0.0	0.0	0.0	613.86536

****
Solucion: 2
****
Solution ID: 128400
Total Cost: 638.8543

Routes:
route 2464149: 0->10->14->4->1->6->7->0
route 2464152: 0->3->22->21->19->17->0
route 2464154: 0->13->23->0
route 2463721: 0->8->9->12->5->11->0
route 2433297: 0->2->15->16->20->24->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2464149	97.1714	0.0	0.0	0.0	0.0	97.1714
2464152	173.2489	0.0	0.0	0.0	0.0	173.2489
2464154	79.0312	0.0	0.0	0.0	0.0	79.0312
2463721	124.2241	0.0	0.0	0.0	0.0	124.2241
2433297	165.1787	0.0	0.0	0.0	0.0	165.1787
Total	638.85425	0.0	0.0	0.0	0.0	638.85425

****
Solucion: 3
****
Solution ID: 126728
Total Cost: 679.3175

Routes:
route 2431879: 0->2->15->5->12->9->0
route 2431880: 0->3->22->21->19->17->0
route 2431883: 0->7->6->14->4->1->10->0
route 2431885: 0->8->11->0
route 2431895: 0->16->20->24->18->23->13->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2431879	156.7141	0.0	0.0	0.0	0.0	156.7141
2431880	173.2489	0.0	0.0	0.0	0.0	173.2489
2431883	131.2658	0.0	0.0	0.0	0.0	131.2658
2431885	28.1078	0.0	0.0	0.0	0.0	28.1078
2431895	189.9809	0.0	0.0	0.0	0.0	189.9809
Total	679.3175	0.0	0.0	0.0	0.0	679.3175

Prueba 3

Ajuste Zonas naturales = 1

*****
Solucion: 0
*****
Solution ID: 2053
Total Cost: 592.1529

Routes:
route 1805: 0->17->24->20->16->15->0
route 270: 0->2->5->12->9->1->6->7->0
route 39769: 0->3->0
route 3824: 0->14->4->10->8->11->0
route 19525: 0->18->19->21->22->23->13->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|1805 |170.1064|0.0 |0.0 |12.0 |0.6 |170.7064|
|270 |136.6417|0.0 |0.0 |7.0 |0.35 |136.9917|
|39769 |10.7704 |0.0 |0.0 |0.0 |0.0 |10.7704 |
|3824 |71.445 |0.0 |0.0 |0.0 |0.0 |71.445 |
|19525 |194.7394|0.0 |0.0 |15.0 |7.5 |202.2394|
|Total |583.7029|0.0 |0.0 |34.0 |8.45 |592.1529|

*****
Solucion: 1
*****
Solution ID: 88
Total Cost: 607.7983

Routes:
route 1699: 0->3->22->21->19->17->0
route 393: 0->18->24->20->16->15->5->2->0
route 1714: 0->9->12->1->6->7->0
route 1716: 0->11->0
route 148: 0->8->10->14->4->13->23->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|1699 |173.2489|0.0 |0.0 |15.0 |7.5 |180.7489|
|393 |180.7948|0.0 |0.0 |12.0 |0.6 |181.3948|
|1714 |114.3963|0.0 |0.0 |0.0 |0.0 |114.3963|
|1716 |12.8062 |0.0 |0.0 |0.0 |0.0 |12.8062 |
|148 |118.4521|0.0 |0.0 |0.0 |0.0 |118.4521|
|Total |599.6983|0.0 |0.0 |27.0 |8.1 |607.79834|

*****
Solucion: 2
*****
Solution ID: 3
Total Cost: 628.0349

Routes:
route 2: 0->2->5->9->12->1->6->7->0
route 3: 0->3->17->16->18->0
route 8: 0->8->10->4->14->13->23->0
route 11: 0->11->0
route 15: 0->15->20->24->19->21->22->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2	139.045	0.0	0.0	0.0	0.0	139.045
3	100.9302	0.0	0.0	0.0	0.0	100.9302
8	120.5449	0.0	0.0	0.0	0.0	120.5449
11	12.8062	0.0	0.0	0.0	0.0	12.8062
15	247.2086	0.0	0.0	15.0	7.5	254.7086
Total	620.5349	0.0	0.0	15.0	7.5	628.0349

Ajuste Zonas naturales = 3

****
Solucion: 0
****
Solution ID: 37086
Total Cost: 607.9108

Routes:
route 720334: 0->3->0
route 720335: 0->15->16->20->24->17->0
route 720345: 0->18->21->19->22->23->13->0
route 720244: 0->2->5->12->9->1->6->7->0
route 720250: 0->14->4->10->8->11->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
720334	10.7704	0.0	0.0	0.0	0.0	10.7704
720335	170.1064	0.0	0.0	12.0	0.6	170.7064
720345	216.0973	0.0	0.0	0.0	0.0	216.0973
720244	136.6417	0.0	0.0	7.0	0.35	136.9917
720250	71.445	0.0	0.0	0.0	0.0	71.445
Total	605.06085	0.0	0.0	19.0	0.95000005	606.0108

****
Solucion: 1
****
Solution ID: 40127
Total Cost: 608.7929

Routes:
route 779717: 0->11->0
route 779722: 0->7->9->12->5->15->2->0
route 722347: 0->10->14->4->1->6->8->0

route 720258: 0->21->19->24->20->16->18->0
route 777298: 0->3->13->23->22->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
779717	12.8062	0.0	0.0	0.0	0.0	12.8062
779722	157.898	0.0	0.0	0.0	0.0	157.898
722347	97.3134	0.0	0.0	0.0	0.0	97.3134
720258	215.0802	0.0	0.0	0.0	0.0	215.0802
777298	125.6951	0.0	0.0	0.0	0.0	125.6951
Total	608.79285	0.0	0.0	0.0	0.0	608.79285

****
Solucion: 2
****
Solution ID: 39684
Total Cost: 615.3163

Routes:
route 770873: 0->2->15->5->12->9->10->0
route 770880: 0->11->0
route 756292: 0->14->4->1->6->7->8->0
route 770886: 0->3->13->23->22->17->0
route 720258: 0->21->19->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
770873	164.7697	0.0	0.0	0.0	0.0	164.7697
770880	12.8062	0.0	0.0	0.0	0.0	12.8062
756292	96.9651	0.0	0.0	0.0	0.0	96.9651
770886	125.6951	0.0	0.0	0.0	0.0	125.6951
720258	215.0802	0.0	0.0	0.0	0.0	215.0802
Total	615.3163	0.0	0.0	0.0	0.0	615.3163

****
Solucion: 3
****
Solution ID: 37659
Total Cost: 621.4751

Routes:
route 731539: 0->18->24->20->16->15->5->2->0
route 731544: 0->11->0
route 731548: 0->9->12->1->6->7->0
route 720357: 0->3->22->19->21->17->0
route 724063: 0->8->10->14->4->13->23->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
731539	180.7948	0.0	0.0	12.0	0.6	181.3948
731544	12.8062	0.0	0.0	0.0	0.0	12.8062
731548	114.3963	0.0	0.0	0.0	0.0	114.3963
720357	193.2257	0.0	0.0	0.0	0.0	193.2257
724063	118.4521	0.0	0.0	0.0	0.0	118.4521

Total	619.6751	0.0	0.0	12.0	0.6	620.2751

****
Solucion: 4
****
Solution ID: 61103
Total Cost: 626.6888

Routes:
route 1188989: 0->16->15->5->2->11->0
route 1188991: 0->3->0
route 873459: 0->7->10->14->4->13->23->22->0
route 727425: 0->9->12->1->6->8->0
route 1185976: 0->18->20->24->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
1188989	126.3234	0.0	0.0	12.0	0.6	126.9234
1188991	10.7704	0.0	0.0	0.0	0.0	10.7704
873459	160.0227	0.0	0.0	0.0	0.0	160.0227
727425	114.5383	0.0	0.0	0.0	0.0	114.5383
1185976	213.234	0.0	0.0	0.0	0.0	213.234
Total	624.8888	0.0	0.0	12.0	0.6	625.4888

****
Solucion: 5
****
Solution ID: 44910
Total Cost: 635.8832

Routes:
route 873177: 0->16->15->5->2->11->0
route 873179: 0->3->0
route 762094: 0->7->9->12->1->6->14->10->0
route 873188: 0->22->23->13->4->8->0
route 725768: 0->18->20->24->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
873177	126.3234	0.0	0.0	12.0	0.6	126.9234
873179	10.7704	0.0	0.0	0.0	0.0	10.7704
762094	130.3751	0.0	0.0	0.0	0.0	130.3751
873188	153.3803	0.0	0.0	0.0	0.0	153.3803
725768	213.234	0.0	0.0	0.0	0.0	213.234
Total	634.0832	0.0	0.0	12.0	0.6	634.6832

****
Solucion: 6
****
Solution ID: 37067
Total Cost: 643.0349

Routes:

route 720023: 0->2->5->9->12->1->6->7->0
route 720024: 0->3->17->16->18->0
route 720029: 0->8->10->4->14->13->23->0
route 720032: 0->11->0
route 720036: 0->15->20->24->19->21->22->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
720023	139.045	0.0	0.0	0.0	0.0	139.045
720024	100.9302	0.0	0.0	0.0	0.0	100.9302
720029	120.5449	0.0	0.0	0.0	0.0	120.5449
720032	12.8062	0.0	0.0	0.0	0.0	12.8062
720036	247.2086	0.0	0.0	15.0	7.5	254.7086
Total	620.5349	0.0	0.0	15.0	7.5	628.0349

Ajuste Zonas naturales = 5

****
Solucion: 0
****
Solution ID: 30821
Total Cost: 608.7929

Routes:
route 594561: 0->7->9->12->5->15->2->0
route 591065: 0->3->13->23->22->17->0
route 594565: 0->10->14->4->1->6->8->0
route 594567: 0->11->0
route 591441: 0->21->19->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
594561	157.898	0.0	0.0	0.0	0.0	157.898
591065	125.6951	0.0	0.0	0.0	0.0	125.6951
594565	97.3134	0.0	0.0	0.0	0.0	97.3134
594567	12.8062	0.0	0.0	0.0	0.0	12.8062
591441	215.0802	0.0	0.0	0.0	0.0	215.0802
Total	608.7929	0.0	0.0	0.0	0.0	608.7929

****
Solucion: 1
****
Solution ID: 33154
Total Cost: 609.8108

Routes:
route 642265: 0->3->0
route 642266: 0->15->16->20->24->17->0
route 642276: 0->18->21->19->22->23->13->0
route 591427: 0->2->5->12->9->1->6->7->0
route 641890: 0->14->4->10->8->11->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
642265	10.7704	0.0	0.0	0.0	0.0	10.7704
642266	170.1064	0.0	0.0	12.0	0.6	170.7064
642276	216.0973	0.0	0.0	0.0	0.0	216.0973
591427	136.6417	0.0	0.0	7.0	0.35	136.9917
641890	71.445	0.0	0.0	0.0	0.0	71.445
Total	605.06085	0.0	0.0	19.0	0.95000005	606.0108

****
Solucion: 2
****
Solution ID: 31221
Total Cost: 622.6751

Routes:
route 591063: 0->9->12->1->6->7->0
route 602700: 0->11->0
route 590924: 0->2->5->15->16->20->24->18->0
route 600915: 0->3->22->19->21->17->0
route 591253: 0->8->10->14->4->13->23->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
591063	114.3963	0.0	0.0	0.0	0.0	114.3963
602700	12.8062	0.0	0.0	0.0	0.0	12.8062
590924	180.7948	0.0	0.0	12.0	0.6	181.3948
600915	193.2257	0.0	0.0	0.0	0.0	193.2257
591253	118.4521	0.0	0.0	0.0	0.0	118.4521
Total	619.6751	0.0	0.0	12.0	0.6	620.2751

****
Solucion: 3
****
Solution ID: 30651
Total Cost: 629.7249

Routes:
route 590829: 0->9->12->1->6->7->0
route 590830: 0->15->5->2->11->0
route 590831: 0->3->13->23->22->17->0
route 590836: 0->8->10->4->14->0
route 590844: 0->21->19->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
590829	114.3963	0.0	0.0	0.0	0.0	114.3963
590830	107.8067	0.0	0.0	0.0	0.0	107.8067
590831	125.6951	0.0	0.0	0.0	0.0	125.6951
590836	66.7466	0.0	0.0	0.0	0.0	66.7466
590844	215.0802	0.0	0.0	0.0	0.0	215.0802
Total	629.7249	0.0	0.0	0.0	0.0	629.7249

Prueba 4

Ajuste de poblaciones de paso = 1, Ajuste Zonas naturales = 1

****

Solucion: 0

****

Solution ID: 1532

Total Cost: 599.436

Routes:

route 29065: 0->3->0

route 29070: 0->10->14->4->8->11->0

route 29080: 0->15->16->20->24->17->0

route 48: 0->2->5->12->9->1->6->7->0

route 28167: 0->18->19->21->22->23->13->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
29065	10.7704	0.0	0.0	0.0	0.0	10.7704
29070	71.7281	0.0	0.0	0.0	0.0	71.7281
29080	170.1064	0.0	0.0	12.0	0.6	170.7064
48	136.6417	850.0	3.0	7.0	0.35	139.9917
28167	194.7394	1000.0	4.0	15.0	7.5	206.2394
Total	583.98596	1850.0	7.0	34.0	8.45	599.436

****

Solucion: 1

****

Solution ID: 50495

Total Cost: 607.536

Routes:

route 3242: 0->10->14->4->1->6->8->0

route 27709: 0->7->9->12->5->15->2->0

route 27713: 0->11->0

route 28162: 0->3->17->24->20->16->0

route 28167: 0->18->19->21->22->23->13->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3242	97.3134	0.0	0.0	0.0	0.0	97.3134
27709	157.898	0.0	0.0	0.0	0.0	157.898
27713	12.8062	0.0	0.0	0.0	0.0	12.8062
28162	133.279	0.0	0.0	0.0	0.0	133.279
28167	194.7394	1000.0	4.0	15.0	7.5	206.2394
Total	596.036	1000.0	4.0	15.0	7.5	607.536

****

Solucion: 2

****

Solution ID: 122

Total Cost: 610.7983

Routes:

route 2481: 0->18->24->20->16->15->5->2->0
route 258: 0->9->12->1->6->7->0
route 2487: 0->11->0
route 54: 0->8->10->14->4->13->23->0
route 1167: 0->3->22->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2481	180.7948	850.0	3.0	12.0	0.6	184.3948
258	114.3963	0.0	0.0	0.0	0.0	114.3963
2487	12.8062	0.0	0.0	0.0	0.0	12.8062
54	118.4521	0.0	0.0	0.0	0.0	118.4521
1167	173.2489	0.0	0.0	15.0	7.5	180.7489
Total	599.6983	850.0	3.0	27.0	8.1	610.79834

****
Solucion: 3
****
Solution ID: 39935
Total Cost: 611.1657

Routes:
route 3242: 0->10->14->4->1->6->8->0
route 766505: 0->3->11->0
route 3427: 0->19->21->22->23->13->0
route 765320: 0->7->9->12->5->15->2->0
route 1356: 0->17->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
3242	97.3134	0.0	0.0	0.0	0.0	97.3134
766505	23.4502	0.0	0.0	0.0	0.0	23.4502
3427	189.9729	0.0	0.0	15.0	7.5	197.4729
765320	157.898	0.0	0.0	0.0	0.0	157.898
1356	135.0312	0.0	0.0	0.0	0.0	135.0312
Total	603.6657	0.0	0.0	15.0	7.5	611.1657

****
Solucion: 4
****
Solution ID: 3
Total Cost: 656.7847

Routes:
route 2: 0->2->5->9->12->1->6->0
route 3: 0->3->23->13->0
route 7: 0->10->4->14->7->8->11->0
route 15: 0->15->20->24->19->21->22->0
route 16: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2	139.0425	850.0	3.0	0.0	0.0	142.0425
3	79.7286	0.0	0.0	0.0	0.0	79.7286

7	76.4931	0.0	0.0	0.0	0.0	76.4931
15	247.2086	1000.0	4.0	15.0	7.5	258.70862
16	99.8119	0.0	0.0	0.0	0.0	99.8119
Total	642.28467	1850.0	7.0	15.0	7.5	656.78467

Ajuste de poblaciones de paso = 1, Ajuste Zonas naturales = 3

****

Solucion: 0

****

Solution ID: 144228

Total Cost: 609.3896

Routes:

route 2792023: 0->11->0

route 2792024: 0->10->14->4->1->6->8->0

route 2792035: 0->7->9->12->5->15->2->0

route 2791691: 0->19->21->24->20->16->18->0

route 2788703: 0->3->13->23->22->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2792023	12.8062	0.0	0.0	0.0	0.0	12.8062
2792024	97.3134	0.0	0.0	0.0	0.0	97.3134
2792035	157.898	0.0	0.0	0.0	0.0	157.898
2791691	215.6769	0.0	0.0	0.0	0.0	215.6769
2788703	125.6951	0.0	0.0	0.0	0.0	125.6951
Total	609.3895	0.0	0.0	0.0	0.0	609.3895

****

Solucion: 1

****

Solution ID: 145962

Total Cost: 611.1939

Routes:

route 2788042: 0->17->24->20->16->15->0

route 2792926: 0->2->5->12->9->1->6->7->0

route 2789697: 0->10->14->4->8->11->0

route 2788958: 0->18->21->19->22->23->13->0

route 2824690: 0->3->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2788042	170.1064	0.0	0.0	12.0	0.6	170.7064
2792926	136.6417	850.0	3.0	7.0	0.35	139.9917
2789697	71.7281	0.0	0.0	0.0	0.0	71.7281
2788958	216.0973	0.0	0.0	0.0	0.0	216.0973
2824690	10.7704	0.0	0.0	0.0	0.0	10.7704

|Total |605.3439|850.0 |3.0 |19.0 |0.95000005|609.2939|

****
Solucion: 2
****
Solution ID: 144854
Total Cost: 624.4751

Routes:
route 2803944: 0->18->24->20->16->15->5->2->0
route 2792725: 0->9->12->1->6->7->0
route 2803950: 0->11->0
route 2797900: 0->3->22->19->21->17->0
route 2792000: 0->8->10->14->4->13->23->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2803944	180.7948	850.0	3.0	12.0	0.6	184.3948
2792725	114.3963	0.0	0.0	0.0	0.0	114.3963
2803950	12.8062	0.0	0.0	0.0	0.0	12.8062
2797900	193.2257	0.0	0.0	0.0	0.0	193.2257
2792000	118.4521	0.0	0.0	0.0	0.0	118.4521
Total	619.6751	850.0	3.0	12.0	0.6	623.2751

****
Solucion: 3
****
Solution ID: 144031
Total Cost: 664.1689

Routes:
route 2787947: 0->2->5->9->12->1->6->0
route 2787948: 0->3->23->13->0
route 2787952: 0->10->4->14->7->8->11->0
route 2787960: 0->15->20->24->21->19->22->0
route 2787961: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
2787947	139.0425	850.0	3.0	0.0	0.0	142.0425
2787948	79.7286	0.0	0.0	0.0	0.0	79.7286
2787952	76.4931	0.0	0.0	0.0	0.0	76.4931
2787960	266.0928	0.0	0.0	0.0	0.0	266.0928
2787961	99.8119	0.0	0.0	0.0	0.0	99.8119
Total	661.1689	850.0	3.0	0.0	0.0	664.1689

Ajuste de poblaciones de paso = 1, Ajuste Zonas naturales = 5

****

Solucion: 0

****

Solution ID: 212082

Total Cost: 609.3896

Routes:

route 4127663: 0->19->21->24->20->16->18->0

route 4130963: 0->10->14->4->1->6->8->0

route 4131402: 0->3->13->23->22->17->0

route 4128690: 0->7->9->12->5->15->2->0

route 4128697: 0->11->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
4127663	215.6769	0.0	0.0	0.0	0.0	215.6769
4130963	97.3134	0.0	0.0	0.0	0.0	97.3134
4131402	125.6951	0.0	0.0	0.0	0.0	125.6951
4128690	157.898	0.0	0.0	0.0	0.0	157.898
4128697	12.8062	0.0	0.0	0.0	0.0	12.8062
Total	609.3896	0.0	0.0	0.0	0.0	609.3896

****

Solucion: 1

****

Solution ID: 218415

Total Cost: 613.0939

Routes:

route 4258446: 0->2->5->12->9->1->6->7->0

route 4128594: 0->10->14->4->8->11->0

route 4127569: 0->15->16->20->24->17->0

route 4256964: 0->3->0

route 4133185: 0->18->21->19->22->23->13->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
4258446	136.6417	850.0	3.0	7.0	0.35	139.9917
4128594	71.7281	0.0	0.0	0.0	0.0	71.7281
4127569	170.1064	0.0	0.0	12.0	0.6	170.7064
4256964	10.7704	0.0	0.0	0.0	0.0	10.7704
4133185	216.0973	0.0	0.0	0.0	0.0	216.0973
Total	605.3439	850.0	3.0	19.0	0.95000005	609.2939

****

Solucion: 2

****

Solution ID: 221776

Total Cost: 625.6751

Routes:

route 4324712: 0->3->22->19->21->17->0
route 4164543: 0->18->24->20->16->15->5->2->0
route 4130590: 0->9->12->1->6->7->0
route 4324729: 0->11->0
route 4136934: 0->8->10->14->4->13->23->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
4324712	193.2257	0.0	0.0	0.0	0.0	193.2257
4164543	180.7948	850.0	3.0	12.0	0.6	184.3948
4130590	114.3963	0.0	0.0	0.0	0.0	114.3963
4324729	12.8062	0.0	0.0	0.0	0.0	12.8062
4136934	118.4521	0.0	0.0	0.0	0.0	118.4521
Total	619.6751	850.0	3.0	12.0	0.6	623.2751

****
Solucion: 3
****
Solution ID: 220838
Total Cost: 630.8888

Routes:
route 4306632: 0->16->15->5->2->11->0
route 4306634: 0->3->0
route 4128484: 0->9->12->1->6->8->0
route 4232840: 0->7->10->14->4->13->23->22->0
route 4303740: 0->18->20->24->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
4306632	126.3234	850.0	3.0	12.0	0.6	129.9234
4306634	10.7704	0.0	0.0	0.0	0.0	10.7704
4128484	114.5383	0.0	0.0	0.0	0.0	114.5383
4232840	160.0227	0.0	0.0	0.0	0.0	160.0227
4303740	213.234	0.0	0.0	0.0	0.0	213.234
Total	624.8888	850.0	3.0	12.0	0.6	628.48883

****
Solucion: 4
****
Solution ID: 211898
Total Cost: 664.1689

Routes:
route 4127325: 0->2->5->9->12->1->6->0
route 4127326: 0->3->23->13->0
route 4127330: 0->10->4->14->7->8->11->0
route 4127338: 0->15->20->24->21->19->22->0
route 4127339: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
4127325	139.0425	850.0	3.0	0.0	0.0	142.0425
4127326	79.7286	0.0	0.0	0.0	0.0	79.7286

|4127330 |76.4931 |0.0 |0.0 |0.0 |0.0 |76.4931 |
|4127338 |266.0928|0.0 |0.0 |0.0 |0.0 |266.0928|
|4127339 |99.8119 |0.0 |0.0 |0.0 |0.0 |99.8119 |
|Total |661.1689|850.0 |3.0 |0.0 |0.0 |664.1689|

Ajuste de poblaciones de paso = 2, Ajuste Zonas naturales = 1

*****
Solucion: 0
*****
Solution ID: 281769
Total Cost: 606.436

Routes:
route 5508446: 0->7->6->1->9->12->5->2->0
route 5508878: 0->18->19->21->22->23->13->0
route 5508443: 0->10->14->4->8->11->0
route 5508885: 0->3->0
route 5507012: 0->15->16->20->24->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|5508446 |136.6417|850.0 |3.0 |7.0 |0.35 |139.9917|
|5508878 |194.7394|1000.0 |4.0 |15.0 |7.5 |206.2394|
|5508443 |71.7281 |0.0 |0.0 |0.0 |0.0 |71.7281 |
|5508885 |10.7704 |0.0 |0.0 |0.0 |0.0 |10.7704 |
|5507012 |170.1064|0.0 |0.0 |12.0 |0.6 |170.7064|
|Total |583.98596|1850.0 |7.0 |34.0 |8.45 |599.436 |

*****
Solucion: 1
*****
Solution ID: 285095
Total Cost: 609.3896

Routes:
route 5572444: 0->19->21->24->20->16->18->0
route 5572453: 0->7->9->12->5->15->2->0
route 5517815: 0->10->14->4->1->6->8->0
route 5572462: 0->11->0
route 5505856: 0->3->13->23->22->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|5572444 |215.6769|0.0 |0.0 |0.0 |0.0 |215.6769|
|5572453 |157.898 |0.0 |0.0 |0.0 |0.0 |157.898 |
|5517815 |97.3134 |0.0 |0.0 |0.0 |0.0 |97.3134 |
|5572462 |12.8062 |0.0 |0.0 |0.0 |0.0 |12.8062 |
|5505856 |125.6951|0.0 |0.0 |0.0 |0.0 |125.6951|
|Total |609.38965|0.0 |0.0 |0.0 |0.0 |609.38965|

*****
Solucion: 2
*****
Solution ID: 283477
Total Cost: 613.5577

Routes:
route 5516014: 0->5->12->9->1->6->7->0
route 5541299: 0->11->0
route 5516228: 0->18->24->20->16->15->2->0
route 5538706: 0->8->10->14->4->13->23->0
route 5509402: 0->3->22->21->19->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|5516014 |135.4218|0.0 |0.0 |7.0 |0.35 |135.7718|
|5541299 |12.8062 |0.0 |0.0 |0.0 |0.0 |12.8062 |
|5516228 |165.1787|0.0 |0.0 |12.0 |0.6 |165.7787|
|5538706 |118.4521|0.0 |0.0 |0.0 |0.0 |118.4521|
|5509402 |173.2489|0.0 |0.0 |15.0 |7.5 |180.7489|
|Total |605.10767|0.0 |0.0 |34.0 |8.45 |613.5577|

*****
Solucion: 3
*****
Solution ID: 281611
Total Cost: 663.7847

Routes:
route 5505498: 0->2->5->9->12->1->6->0
route 5505499: 0->3->23->13->0
route 5505503: 0->10->4->14->7->8->11->0
route 5505511: 0->15->20->24->19->21->22->0
route 5505512: 0->18->16->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|5505498 |139.0425|850.0 |3.0 |0.0 |0.0 |142.0425|
|5505499 |79.7286 |0.0 |0.0 |0.0 |0.0 |79.7286 |
|5505503 |76.4931 |0.0 |0.0 |0.0 |0.0 |76.4931 |
|5505511 |247.2086|1000.0 |4.0 |15.0 |7.5 |258.70862|
|5505512 |99.8119 |0.0 |0.0 |0.0 |0.0 |99.8119 |
|Total |642.28467|1850.0 |7.0 |15.0 |7.5 |656.78467|

Ajuste de poblaciones de paso = 2, Ajuste Zonas naturales = 2

****

Solucion: 0

****

Solution ID: 348509

Total Cost: 609.3896

Routes:

route 6782669: 0->19->21->24->20->16->18->0

route 6811587: 0->11->0

route 6811588: 0->10->14->4->1->6->8->0

route 6811591: 0->7->9->12->5->15->2->0

route 6810168: 0->3->13->23->22->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
6782669	215.6769	0.0	0.0	0.0	0.0	215.6769
6811587	12.8062	0.0	0.0	0.0	0.0	12.8062
6811588	97.3134	0.0	0.0	0.0	0.0	97.3134
6811591	157.898	0.0	0.0	0.0	0.0	157.898
6810168	125.6951	0.0	0.0	0.0	0.0	125.6951
Total	609.38965	0.0	0.0	0.0	0.0	609.38965

****

Solucion: 1

****

Solution ID: 349017

Total Cost: 613.2439

Routes:

route 6821796: 0->3->0

route 6789744: 0->2->5->12->9->1->6->7->0

route 6795597: 0->10->14->4->8->11->0

route 6816638: 0->18->21->19->22->23->13->0

route 6798064: 0->15->16->20->24->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
6821796	10.7704	0.0	0.0	0.0	0.0	10.7704
6789744	136.6417	850.0	3.0	7.0	0.35	139.9917
6795597	71.7281	0.0	0.0	0.0	0.0	71.7281
6816638	216.0973	0.0	0.0	0.0	0.0	216.0973
6798064	170.1064	0.0	0.0	12.0	0.6	170.7064
Total	605.3439	850.0	3.0	19.0	0.95000005	609.29395

****

Solucion: 2

****

Solution ID: 347128

Total Cost: 621.8983

Routes:

route 6784149: 0->18->24->20->16->15->5->2->0
route 6785140: 0->9->12->1->6->7->0
route 6785142: 0->11->0
route 6784798: 0->8->10->14->4->13->23->0
route 6782589: 0->3->22->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
6784149	180.7948	850.0	3.0	12.0	0.6	184.3948
6785140	114.3963	0.0	0.0	0.0	0.0	114.3963
6785142	12.8062	0.0	0.0	0.0	0.0	12.8062
6784798	118.4521	0.0	0.0	0.0	0.0	118.4521
6782589	173.2489	0.0	0.0	15.0	7.5	180.7489
Total	599.6983	850.0	3.0	27.0	8.1	610.79834

****
Solucion: 3
****
Solution ID: 399892
Total Cost: 632.0888

Routes:
route 7816755: 0->9->12->1->6->8->0
route 6856890: 0->16->15->5->2->11->0
route 6880197: 0->7->10->14->4->13->23->22->0
route 7816766: 0->3->0
route 6801845: 0->18->20->24->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
7816755	114.5383	0.0	0.0	0.0	0.0	114.5383
6856890	126.3234	850.0	3.0	12.0	0.6	129.9234
6880197	160.0227	0.0	0.0	0.0	0.0	160.0227
7816766	10.7704	0.0	0.0	0.0	0.0	10.7704
6801845	213.234	0.0	0.0	0.0	0.0	213.234
Total	624.8888	850.0	3.0	12.0	0.6	628.4888

****
Solucion: 4
****
Solution ID: 346995
Total Cost: 667.1689

Routes:
route 6782497: 0->2->5->9->12->1->6->0
route 6782498: 0->3->23->13->0
route 6782502: 0->10->4->14->7->8->11->0
route 6782510: 0->15->20->24->21->19->22->0
route 6782511: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
6782497	139.0425	850.0	3.0	0.0	0.0	142.0425
6782498	79.7286	0.0	0.0	0.0	0.0	79.7286

6782502	76.4931	0.0	0.0	0.0	0.0	76.4931
6782510	266.0928	0.0	0.0	0.0	0.0	266.0928
6782511	99.8119	0.0	0.0	0.0	0.0	99.8119
Total	661.1689	850.0	3.0	0.0	0.0	664.1689

Ajuste de poblaciones de paso = 2, Ajuste Zonas naturales = 3

****

Solucion: 0

****

Solution ID: 411669

Total Cost: 609.3896

Routes:

route 8047803: 0->7->9->12->5->15->2->0

route 8047808: 0->10->14->4->1->6->8->0

route 8047809: 0->11->0

route 8047475: 0->19->21->24->20->16->18->0

route 8045453: 0->3->13->23->22->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
8047803	157.898	0.0	0.0	0.0	0.0	157.898
8047808	97.3134	0.0	0.0	0.0	0.0	97.3134
8047809	12.8062	0.0	0.0	0.0	0.0	12.8062
8047475	215.6769	0.0	0.0	0.0	0.0	215.6769
8045453	125.6951	0.0	0.0	0.0	0.0	125.6951
Total	609.38965	0.0	0.0	0.0	0.0	609.38965

****

Solucion: 1

****

Solution ID: 413520

Total Cost: 614.1939

Routes:

route 8084939: 0->3->0

route 8047628: 0->2->5->12->9->1->6->7->0

route 8053377: 0->11->8->4->14->10->0

route 8051919: 0->18->21->19->22->23->13->0

route 8082487: 0->17->24->20->16->15->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
8084939	10.7704	0.0	0.0	0.0	0.0	10.7704
8047628	136.6417	850.0	3.0	7.0	0.35	139.9917
8053377	71.7281	0.0	0.0	0.0	0.0	71.7281
8051919	216.0973	0.0	0.0	0.0	0.0	216.0973
8082487	170.1064	0.0	0.0	12.0	0.6	170.7064
Total	605.3439	850.0	3.0	19.0	0.95000005	609.29395

*****
Solucion: 2
*****
Solution ID: 412829
Total Cost: 627.4751

Routes:
route 8047171: 0->9->12->1->6->7->0
route 8071007: 0->11->0
route 8046751: 0->18->24->20->16->15->5->2->0
route 8068160: 0->8->10->14->4->13->23->0
route 8049540: 0->3->22->19->21->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|8047171 |114.3963|0.0 |0.0 |0.0 |0.0 |114.3963|
|8071007 |12.8062 |0.0 |0.0 |0.0 |0.0 |12.8062 |
|8046751 |180.7948|850.0 |3.0 |12.0 |0.6 |184.3948|
|8068160 |118.4521|0.0 |0.0 |0.0 |0.0 |118.4521|
|8049540 |193.2257|0.0 |0.0 |0.0 |0.0 |193.2257|
|Total |619.6751|850.0 |3.0 |12.0 |0.6 |623.2751|

*****
Solucion: 3
*****
Solution ID: 411547
Total Cost: 667.1689

Routes:
route 8045279: 0->2->5->9->12->1->6->0
route 8045280: 0->3->23->13->0
route 8045284: 0->10->4->14->7->8->11->0
route 8045292: 0->15->20->24->21->19->22->0
route 8045293: 0->18->16->17->0

|Ruta ID |KM |Pob |C.Pob |Reserv |C.Reserv|C.Total |
|8045279 |139.0425|850.0 |3.0 |0.0 |0.0 |142.0425|
|8045280 |79.7286 |0.0 |0.0 |0.0 |0.0 |79.7286 |
|8045284 |76.4931 |0.0 |0.0 |0.0 |0.0 |76.4931 |
|8045292 |266.0928|0.0 |0.0 |0.0 |0.0 |266.0928|
|8045293 |99.8119 |0.0 |0.0 |0.0 |0.0 |99.8119 |
|Total |661.1689|850.0 |3.0 |0.0 |0.0 |664.1689|

Ajuste de poblaciones de paso = 3, Ajuste Zonas naturales = 1

****

Solucion: 0

****

Solution ID: 478389

Total Cost: 609.3896

Routes:

route 9360709: 0->10->14->4->1->6->8->0

route 9360711: 0->2->15->5->12->9->7->0

route 9360718: 0->11->0

route 9355830: 0->3->13->23->22->17->0

route 9355843: 0->19->21->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9360709	97.3134	0.0	0.0	0.0	0.0	97.3134
9360711	157.898	0.0	0.0	0.0	0.0	157.898
9360718	12.8062	0.0	0.0	0.0	0.0	12.8062
9355830	125.6951	0.0	0.0	0.0	0.0	125.6951
9355843	215.6769	0.0	0.0	0.0	0.0	215.6769
Total	609.3896	0.0	0.0	0.0	0.0	609.3896

****

Solucion: 1

****

Solution ID: 479341

Total Cost: 611.1657

Routes:

route 9360711: 0->2->15->5->12->9->7->0

route 9379806: 0->3->11->0

route 9360709: 0->10->14->4->1->6->8->0

route 9356149: 0->13->23->22->21->19->0

route 9359256: 0->17->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9360711	157.898	0.0	0.0	0.0	0.0	157.898
9379806	23.4502	0.0	0.0	0.0	0.0	23.4502
9360709	97.3134	0.0	0.0	0.0	0.0	97.3134
9356149	189.9729	0.0	0.0	15.0	7.5	197.4729
9359256	135.0312	0.0	0.0	0.0	0.0	135.0312
Total	603.6657	0.0	0.0	15.0	7.5	611.1657

****

Solucion: 2

****

Solution ID: 478975

Total Cost: 613.5577

Routes:

route 9360501: 0->5->12->9->1->6->7->0
route 9371776: 0->18->24->20->16->15->2->0
route 9371777: 0->11->0
route 9367132: 0->8->10->14->4->13->23->0
route 9364896: 0->3->22->21->19->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9360501	135.4218	0.0	0.0	7.0	0.35	135.7718
9371776	165.1787	0.0	0.0	12.0	0.6	165.7787
9371777	12.8062	0.0	0.0	0.0	0.0	12.8062
9367132	118.4521	0.0	0.0	0.0	0.0	118.4521
9364896	173.2489	0.0	0.0	15.0	7.5	180.7489
Total	605.1077	0.0	0.0	34.0	8.45	613.55774

****

Solucion: 3

****

Solution ID: 478153
Total Cost: 633.2329

Routes:
route 9355784: 0->9->12->1->6->8->0
route 9355785: 0->2->5->15->16->20->24->18->0
route 9355786: 0->3->22->21->19->17->0
route 9355790: 0->7->14->4->10->13->23->0
route 9355794: 0->11->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9355784	114.5383	0.0	0.0	0.0	0.0	114.5383
9355785	180.7948	850.0	3.0	12.0	0.6	184.3948
9355786	173.2489	0.0	0.0	15.0	7.5	180.7489
9355790	134.7447	0.0	0.0	0.0	0.0	134.7447
9355794	12.8062	0.0	0.0	0.0	0.0	12.8062
Total	616.13293	850.0	3.0	27.0	8.1	627.2329

Ajuste de poblaciones de paso = 3, Ajuste Zonas naturales = 2

****

Solucion: 0

****

Solution ID: 491171
Total Cost: 609.3896

Routes:
route 9617178: 0->19->21->24->20->16->18->0
route 9596845: 0->10->14->4->1->6->8->0
route 9617190: 0->3->13->23->22->17->0
route 9617191: 0->11->0
route 9596850: 0->7->9->12->5->15->2->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9617178	215.6769	0.0	0.0	0.0	0.0	215.6769
9596845	97.3134	0.0	0.0	0.0	0.0	97.3134
9617190	125.6951	0.0	0.0	0.0	0.0	125.6951
9617191	12.8062	0.0	0.0	0.0	0.0	12.8062
9596850	157.898	0.0	0.0	0.0	0.0	157.898
Total	609.3896	0.0	0.0	0.0	0.0	609.3896

****

Solucion: 1

****

Solution ID: 490713

Total Cost: 616.2187

Routes:

route 9603862: 0->2->15->5->12->9->6->7->0

route 9603870: 0->10->14->4->1->8->0

route 9603871: 0->11->0

route 9584774: 0->3->13->23->22->17->0

route 9585338: 0->19->21->24->20->16->18->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9603862	164.3151	0.0	0.0	7.0	0.35	164.6651
9603870	97.0254	0.0	0.0	0.0	0.0	97.0254
9603871	12.8062	0.0	0.0	0.0	0.0	12.8062
9584774	125.6951	0.0	0.0	0.0	0.0	125.6951
9585338	215.6769	0.0	0.0	0.0	0.0	215.6769
Total	615.5187	0.0	0.0	7.0	0.35	615.8687

****

Solucion: 2

****

Solution ID: 489785

Total Cost: 616.2439

Routes:

route 9589879: 0->10->14->4->8->11->0

route 9589884: 0->3->0

route 9585793: 0->18->21->19->22->23->13->0

route 9589708: 0->2->5->12->9->1->6->7->0

route 9587076: 0->17->24->20->16->15->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9589879	71.7281	0.0	0.0	0.0	0.0	71.7281
9589884	10.7704	0.0	0.0	0.0	0.0	10.7704
9585793	216.0973	0.0	0.0	0.0	0.0	216.0973
9589708	136.6417	850.0	3.0	7.0	0.35	139.9917
9587076	170.1064	0.0	0.0	12.0	0.6	170.7064
Total	605.34393	850.0	3.0	19.0	0.95000005	609.29395

****
Solucion: 3
****
Solution ID: 490150
Total Cost: 618.6657

Routes:
route 9596845: 0->10->14->4->1->6->8->0
route 9596850: 0->7->9->12->5->15->2->0
route 9596853: 0->3->11->0
route 9596202: 0->17->24->20->16->18->0
route 9584884: 0->13->23->22->21->19->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9596845	97.3134	0.0	0.0	0.0	0.0	97.3134
9596850	157.898	0.0	0.0	0.0	0.0	157.898
9596853	23.4502	0.0	0.0	0.0	0.0	23.4502
9596202	135.0312	0.0	0.0	0.0	0.0	135.0312
9584884	189.9729	0.0	0.0	15.0	7.5	197.4729
Total	603.6657	0.0	0.0	15.0	7.5	611.1657

****
Solucion: 4
****
Solution ID: 492267
Total Cost: 622.0077

Routes:
route 9638473: 0->5->12->9->1->6->7->0
route 9638476: 0->11->0
route 9638483: 0->18->24->20->16->15->2->0
route 9635367: 0->3->22->21->19->17->0
route 9590080: 0->8->10->14->4->13->23->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9638473	135.4218	0.0	0.0	7.0	0.35	135.7718
9638476	12.8062	0.0	0.0	0.0	0.0	12.8062
9638483	165.1787	0.0	0.0	12.0	0.6	165.7787
9635367	173.2489	0.0	0.0	15.0	7.5	180.7489
9590080	118.4521	0.0	0.0	0.0	0.0	118.4521
Total	605.10767	0.0	0.0	34.0	8.45	613.5577

****
Solucion: 5
****
Solution ID: 489535
Total Cost: 670.1689

Routes:
route 9584683: 0->2->5->9->12->1->6->0
route 9584684: 0->3->23->13->0

route 9584688: 0->10->4->14->7->8->11->0
route 9584696: 0->15->20->24->21->19->22->0
route 9584697: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
9584683	139.0425	850.0	3.0	0.0	0.0	142.0425
9584684	79.7286	0.0	0.0	0.0	0.0	79.7286
9584688	76.4931	0.0	0.0	0.0	0.0	76.4931
9584696	266.0928	0.0	0.0	0.0	0.0	266.0928
9584697	99.8119	0.0	0.0	0.0	0.0	99.8119
Total	661.1689	850.0	3.0	0.0	0.0	664.1689

Ajuste de poblaciones de paso = 3, Ajuste Zonas naturales = 3

****
Solucion: 0
****
Solution ID: 533241
Total Cost: 609.3896

Routes:
route 10451265: 0->19->21->24->20->16->18->0
route 10442400: 0->10->14->4->1->6->8->0
route 10451277: 0->7->9->12->5->15->2->0
route 10451279: 0->11->0
route 10440786: 0->3->13->23->22->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
10451265	215.6769	0.0	0.0	0.0	0.0	215.6769
10442400	97.3134	0.0	0.0	0.0	0.0	97.3134
10451277	157.898	0.0	0.0	0.0	0.0	157.898
10451279	12.8062	0.0	0.0	0.0	0.0	12.8062
10440786	125.6951	0.0	0.0	0.0	0.0	125.6951
Total	609.38965	0.0	0.0	0.0	0.0	609.38965

****
Solucion: 1
****
Solution ID: 540730
Total Cost: 617.1939

Routes:
route 10457328: 0->2->5->12->9->1->6->7->0
route 10600256: 0->3->0
route 10498552: 0->10->14->4->8->11->0
route 10597926: 0->15->16->20->24->17->0
route 10441009: 0->18->21->19->22->23->13->0

```
|Ruta ID |KM      |Pob     |C.Pob   |Reserv  |C.Reserv|C.Total |
|10457328|136.6417|850.0   |3.0     |7.0     |0.35    |139.9917|
|10600256|10.7704 |0.0     |0.0     |0.0     |0.0     |10.7704 |
|10498552|71.7281 |0.0     |0.0     |0.0     |0.0     |71.7281 |
|10597926|170.1064|0.0     |0.0     |12.0    |0.6     |170.7064|
|10441009|216.0973|0.0     |0.0     |0.0     |0.0     |216.0973|
|Total   |605.3439|850.0   |3.0     |19.0    |0.95000005|609.2939|


****************************************************************
Solucion: 2
****************************************************************
Solution ID: 532849
Total Cost: 627.9345

Routes:
route 10443998: 0->11->0
route 10442091: 0->5->12->9->1->6->7->0
route 10444010: 0->8->10->14->4->13->23->0
route 10440933: 0->3->22->19->21->17->0
route 10442076: 0->18->24->20->16->15->2->0


|Ruta ID |KM      |Pob     |C.Pob   |Reserv  |C.Reserv|C.Total |
|10443998|12.8062 |0.0     |0.0     |0.0     |0.0     |12.8062 |
|10442091|135.4218|0.0     |0.0     |7.0     |0.35    |135.7718|
|10444010|118.4521|0.0     |0.0     |0.0     |0.0     |118.4521|
|10440933|193.2257|0.0     |0.0     |0.0     |0.0     |193.2257|
|10442076|165.1787|0.0     |0.0     |12.0    |0.6     |165.7787|
|Total   |625.08453|0.0    |0.0     |19.0    |0.95000005|626.0345|


****************************************************************
Solucion: 3
****************************************************************
Solution ID: 558081
Total Cost: 635.6888

Routes:
route 10943996: 0->16->15->5->2->11->0
route 10943998: 0->3->0
route 10451384: 0->9->12->1->6->8->0
route 10454846: 0->7->10->14->4->13->23->22->0
route 10941399: 0->18->20->24->21->19->17->0


|Ruta ID |KM      |Pob     |C.Pob   |Reserv  |C.Reserv|C.Total |
|10943996|126.3234|850.0   |3.0     |12.0    |0.6     |129.9234|
|10943998|10.7704 |0.0     |0.0     |0.0     |0.0     |10.7704 |
|10451384|114.5383|0.0     |0.0     |0.0     |0.0     |114.5383|
|10454846|160.0227|0.0     |0.0     |0.0     |0.0     |160.0227|
|10941399|213.234 |0.0     |0.0     |0.0     |0.0     |213.234 |
|Total   |624.8888|850.0   |3.0     |12.0    |0.6     |628.48883|


****************************************************************
```

Solucion: 4
****
Solution ID: 532689
Total Cost: 670.1689

Routes:
route 10440510: 0->2->5->9->12->1->6->0
route 10440511: 0->3->23->13->0
route 10440515: 0->10->4->14->7->8->11->0
route 10440523: 0->15->20->24->21->19->22->0
route 10440524: 0->18->16->17->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
10440510	139.0425	850.0	3.0	0.0	0.0	142.0425
10440511	79.7286	0.0	0.0	0.0	0.0	79.7286
10440515	76.4931	0.0	0.0	0.0	0.0	76.4931
10440523	266.0928	0.0	0.0	0.0	0.0	266.0928
10440524	99.8119	0.0	0.0	0.0	0.0	99.8119
Total	661.1689	850.0	3.0	0.0	0.0	664.1689

Ajuste de poblaciones de paso = 10, Ajuste Zonas naturales = 10

****
Solucion: 0
****
Solution ID: 572387
Total Cost: 609.3896

Routes:
route 11220873: 0->19->21->24->20->16->18->0
route 11226457: 0->7->9->12->5->15->2->0
route 11226461: 0->11->0
route 11226133: 0->3->13->23->22->17->0
route 11225257: 0->10->14->4->1->6->8->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
11220873	215.6769	0.0	0.0	0.0	0.0	215.6769
11226457	157.898	0.0	0.0	0.0	0.0	157.898
11226461	12.8062	0.0	0.0	0.0	0.0	12.8062
11226133	125.6951	0.0	0.0	0.0	0.0	125.6951
11225257	97.3134	0.0	0.0	0.0	0.0	97.3134
Total	609.3896	0.0	0.0	0.0	0.0	609.3896

****
Solucion: 1
****
Solution ID: 625620
Total Cost: 639.3359

Routes:
route 11235279: 0->3->22->19->21->17->0
route 12282924: 0->11->0
route 12145309: 0->23->13->4->14->7->8->0
route 11219924: 0->2->15->20->24->16->18->0
route 11594812: 0->5->9->12->1->6->10->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
11235279	193.2257	0.0	0.0	0.0	0.0	193.2257
12282924	12.8062	0.0	0.0	0.0	0.0	12.8062
12145309	123.0534	0.0	0.0	0.0	0.0	123.0534
11219924	168.2692	0.0	0.0	0.0	0.0	168.2692
11594812	141.9814	0.0	0.0	0.0	0.0	141.9814
Total	639.3359	0.0	0.0	0.0	0.0	639.3359

****
Solucion: 2
****
Solution ID: 618582
Total Cost: 652.3352
Routes:
route 12143488: 0->2->15->5->16->18->0
route 11339694: 0->9->12->1->6->8->0
route 12143493: 0->11->0
route 11223451: 0->3->17->19->21->24->20->0
route 11333712: 0->7->10->14->4->13->23->22->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
12143488	151.6851	0.0	0.0	0.0	0.0	151.6851
11339694	114.5383	0.0	0.0	0.0	0.0	114.5383
12143493	12.8062	0.0	0.0	0.0	0.0	12.8062
11223451	213.2829	0.0	0.0	0.0	0.0	213.2829
11333712	160.0227	0.0	0.0	0.0	0.0	160.0227
Total	652.3352	0.0	0.0	0.0	0.0	652.3352

****
Solucion: 3
****
Solution ID: 572034
Total Cost: 729.2399

Routes:
route 11219607: 0->2->9->12->5->8->0
route 11219608: 0->3->23->13->0
route 11219611: 0->7->6->14->4->1->10->0
route 11219616: 0->11->18->16->17->0
route 11219620: 0->15->20->24->21->19->22->0

Ruta ID	KM	Pob	C.Pob	Reserv	C.Reserv	C.Total
11219607	144.2658	0.0	0.0	0.0	0.0	144.2658
11219608	79.7286	0.0	0.0	0.0	0.0	79.7286
11219611	131.2658	0.0	0.0	0.0	0.0	131.2658
11219616	107.8869	0.0	0.0	0.0	0.0	107.8869
11219620	266.0928	0.0	0.0	0.0	0.0	266.0928
Total	729.23987	0.0	0.0	0.0	0.0	729.23987